FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 1

[75.73] Arquitectura de Software
Segundo cuatrimestre de 2022

Mariotti, Franco	102223	fmariotti@fi.uba.ar
Alasino, Franco	100853	falasino@fi.uba.ar
Fusco, Franco Martin	102692	ffusco@fi.uba.ar
Buceta, Belen	102121	bbuceta@fi.uba.ar
Di Como, Juan Pablo	102889	jdicomo@fi.uba.ar

Índice

1. Introducción **2**

2. Seccion 1 **2**
2.1. Components & Connectors . . . 2
2.2. Ping . . . 3
2.2.1. Nodo . . . 3
2.2.2. Nodo replicado . . . 4
2.2.3. Conclusion . . . 5
2.3. Fibonacci n=26 . . . 5
2.3.1. Nodo . . . 6
2.3.2. Nodo replicado . . . 7
2.4. Fibonacci intensivo n=33 . . . 8
2.4.1. Nodo . . . 8
2.4.2. Nodo replicado . . . 9
2.5. Analisis de Bbox-1 . . . 10
2.6. Analisis de Bbox-2 . . . 11

3. Sección 2 **12**
3.1. Sincrónico - Asincrónico . . . 12
3.2. Cantidad de workers . . . 14
3.3. Response Time . . . 14
3.3.1. Response Time Bbox 1 . . . 14
3.3.2. Response Time Bbox 2 . . . 15

4. Sección 3 **15**
4.1. Hipótesis . . . 15
4.2. Modelado de endpoints . . . 16
4.3. Escenarios planteados . . . 16
4.3.1. Escenario numero 1 . . . 16
4.3.2. Escenario numero 2 . . . 17
4.3.3. Escenario numero 3 . . . 17
4.4. Conclusiones . . . 18

1. Introducción

El presente trabajo reune la resolución del primer trabajo práctico de la materia. El cual tiene como objetivo analizar, en base a la información de performance, el rendimiento que un Web Server posee sometiéndolo a diferentes tipos de escenarios.

Contamos con los siguientes endpoints:

- GET /ping
  - Este endpoint recibe una request HTTP y simplemente responde, devolviendo la cadena **ping**
- GET $/fibonacci?n =< n >$
  - Representa un endpoint que realiza un procesamiento extenuente
  - Este endpoint calcula el fibonacci de **n**.
- GET /bbox-1
  - Inicia una comunicacion HTTP con el endpoint **http://bbox:9090/**
- GET /bbox-2
  - Inicia una comunicacion HTTP con el endpoint **http://bbox:9091/**

2. Seccion 1

En esta sección analizaremos la performance sobre distintos endpoints del web server configurado considerando dos tipos de deployments:

- Un solo nodo (un container).
- Un nodo replicado en múltiples containers.

Se crearan una variedad de escenarios de carga donde simularemos llegadas de distintas cantidades de requests a los endpoints, modificando los parámetros como máximos de requests, duración total y arrival rate, entre otros.
Con una herramienta iremos monitoreando estos escenarios, poniendo atención en como van variando en función de las características del escenario en cuestión.

2.1. Components & Connectors

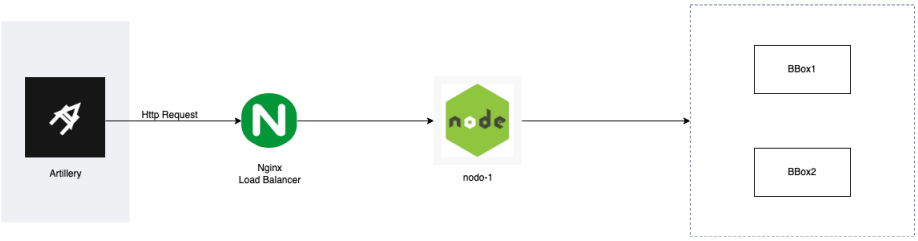


Figura 1: Single Node

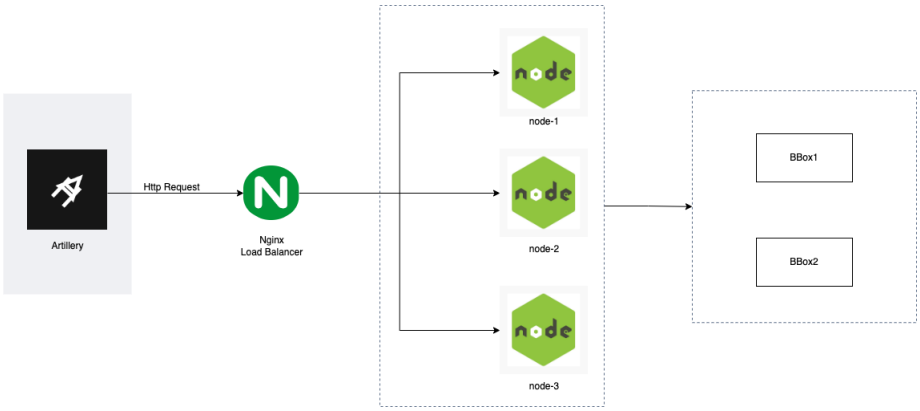


Figura 2: Replicated Node

2.2. Ping

Para este escenario, tenemos las siguientes fases:

- **Ramp**
- **Plain**

2.2.1. Nodo

En el siguiente gráfico, se puede ver como artillery empieza a generar workers para pegarle al enpoint hasta llegar a las 200 requestes por segundo. Tal y como esta definido el escenario, el Ramp dura 30 segundos y nos sirve para ver como maneja el servidor un aumento de requests muy grande en muy poco tiempo, y luego entra en la fase Plain, en la que se queda a 200 req/seg, que dura 120 segundos. El objetivo de esta fase es comparar un estado estable en el cual no hay mucha variación de requests de un momento a otro, con los otros estados como puede ser el ramp.

En el siguiente gráfico observamos el tiempo de respuesta y los recursos utilizados por el servidor. Respecto al response time podemos visualizar que se sufre un pico de aproximadamente 500ms al momento de llegar al pico de requests, esto se debe a que el servidor sufre un aumento muy grande de requests de un momento a otro y por lo tanto tarda en estabilizarse nuevamente. Sobre los resources, se puede observar que al principio el uso de cpu va aumentando hasta llegar a un pico. El aumento se debe a la alta variación de requests/s, una vez llegado al pico, el cpu

no llega al pico de uso. Una vez las requests/s se estabilizan, el uso del cpu va disminuyen. Una particularidad que se observa es que al momento que baja el response time, el uso del cpu aumenta, lo cual tiene sentido ya que está correlacionado con que el uso de cpu: Si el response time disminuye entonces el uso de cpu aumenta.

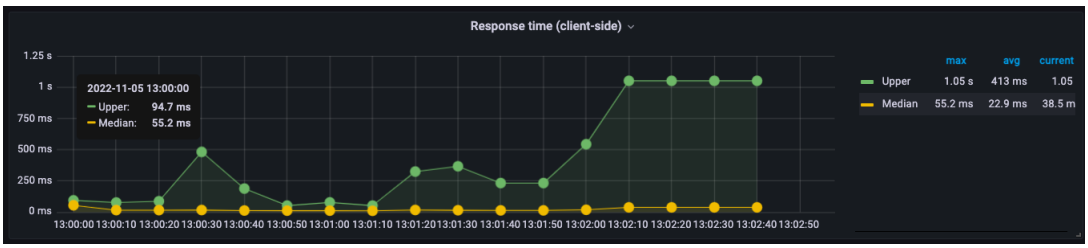


Figura 3: Response time (client)

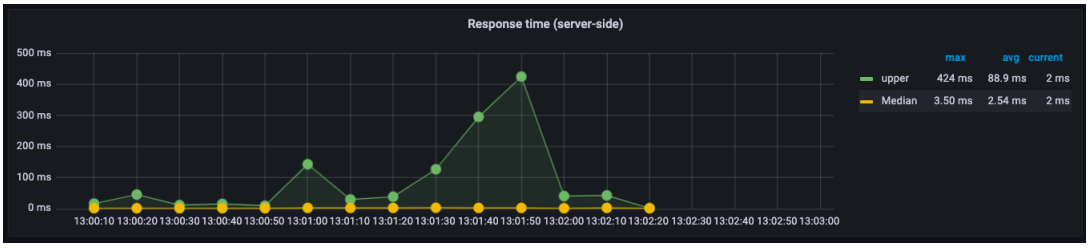


Figura 4: Response time (server)

Una conclusión que podemos sacar observando los gráficos de response time de arriba, es que el nginx parece estar generando cierta latencia, dado que que el response time visto desde el lado del cliente en promedio en mayor al del tiempo percibido por la aplicación Node.

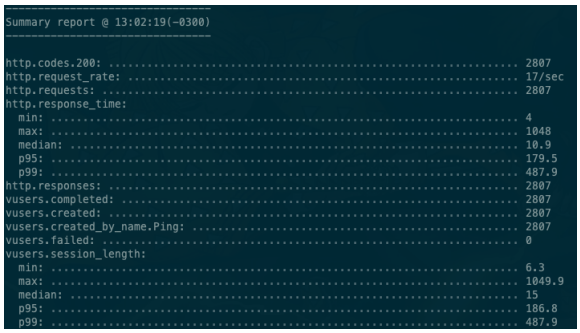


Figura 5: Reponse time(server)

2.2.2. Nodo replicado

En el siguiente gráfico se puede observar como, cuando se tienen nodos replicados y nginx balancea la carga entre ellos, el uso de CPU disminuye debido a que en vez de que un solo nodo realice todo el trabajo por si solo, ahora se tienen mas nodos para distribuir mejor la carga entre ellos. También se puede apreciar un gran aumento en el tiempo de respuesta. Esto es porque nginx tarda en asignar a cada nodo la carga.

Figura 6: Reponse time(server)

En este caso también podemos observar en los gráficos de response time que el tiempo visto desde el cliente es mayor al percibido por la aplicación de Node y que tanto el response time visto desde el client como desde el server, es menor que cuando teníamos un solo nodo.

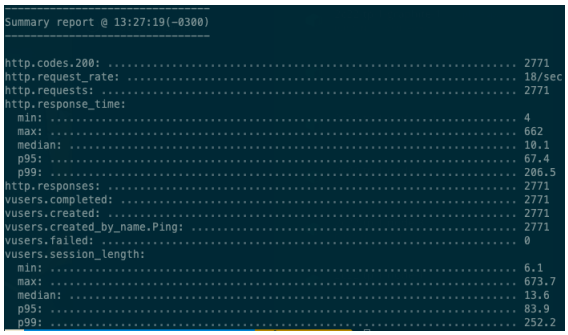


Figura 7: Reponse time(server)

2.2.3. Conclusion

Tener un nginx que funcione como un intermediario en el caso de un nodo, o como load balancer en el caso de multiples nodos, genera latencia en las request del cliente.

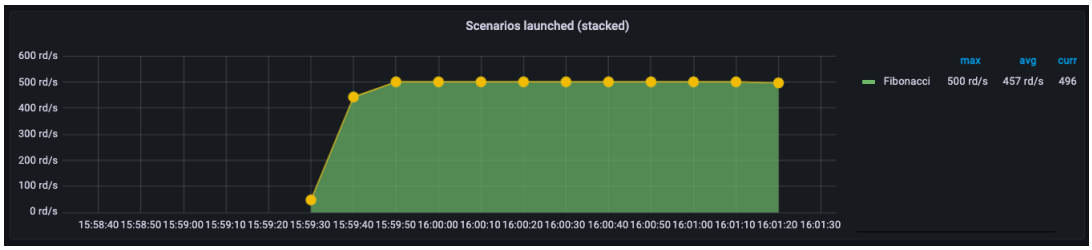
2.3. Fibonacci n=26

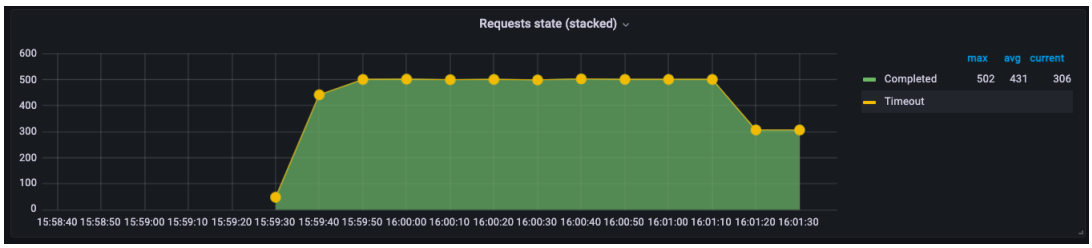
Este escenario se basa en utilizar un endpoint que calcula el fibonacci de un numero de manera recursiva. Esto realiza 2^n operaciones. Para este pŕimer escenario se utilizo un n 'bajo' en relación al que se va a utilizar mas adelante. Para este escenario, tenemos las siguientes fases:

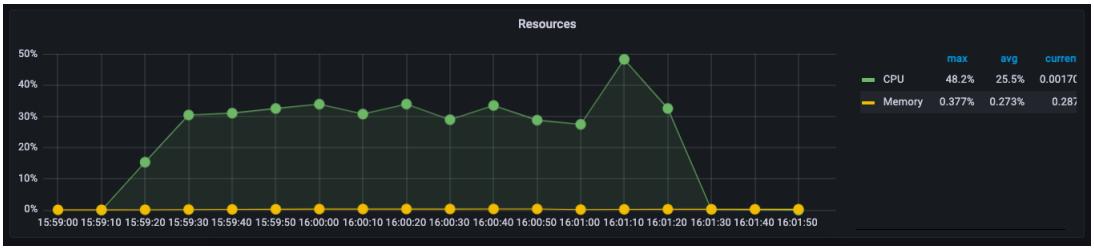
- **Start**
- **Positive Ramp**
- **Plain**
- **Negative Ramp**
- **Final**

2.3.1. Nodo

En el siguiente gráfico se puede observar la definición del escenario.





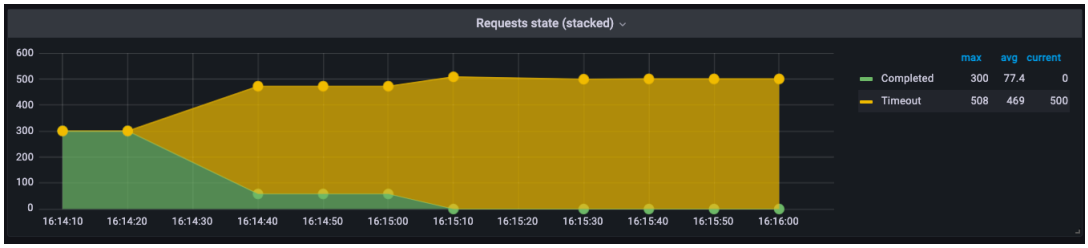


Uno de los objetivos de esta prueba era ver como se comportaba un endpoint que generara mucho CPU load.Decidimos probar con un n relativamente grande y teniendo muchos clientes. Por un lado podemos notar que el consumo de CPU no es cercano al 50 %, pero probando con distintas configuraciones mas intensivas si bien llegamos a un porcentaje de CPU mayor, también comenzaban a producirse errores de timeout. Creemos que quizá sea alguna limitación generada por la VM creada por docker en sistemas macOS.

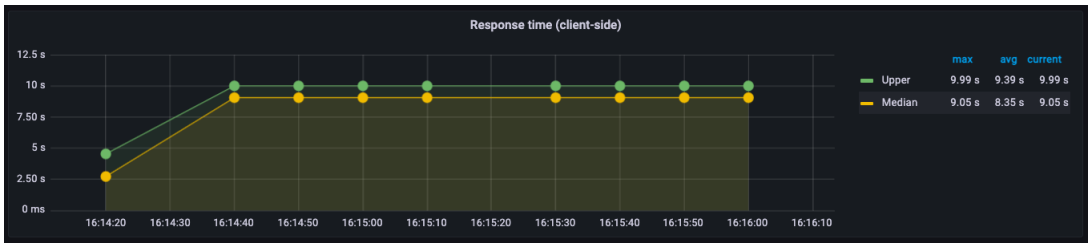
```
Summary report @ 16:01:25(-0300)

http.codes.200: ..................................................... 5859
http.request_rate: .................................................. 44/sec
http.requests: ...................................................... 5859
http.response_time:
  min: .............................................................. 5
  max: .............................................................. 4700
  median: ........................................................... 15
  p95: .............................................................. 3395.5
  p99: .............................................................. 4403.8
http.responses: ..................................................... 5859
vusers.completed: ................................................... 5859
vusers.created: ..................................................... 5859
vusers.created_by_name.Fibonacci: ................................... 5859
vusers.failed: ...................................................... 0
vusers.session_length:
  min: .............................................................. 7.7
  max: .............................................................. 4775.7
  median: ........................................................... 18
  p95: .............................................................. 3395.5
  p99: .............................................................. 4403.8
```

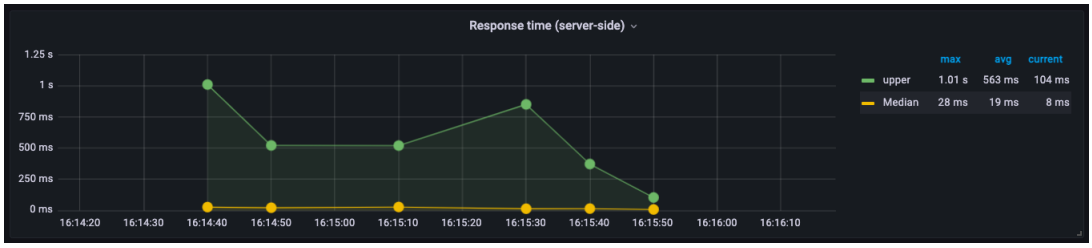
2.3.2. Nodo replicado

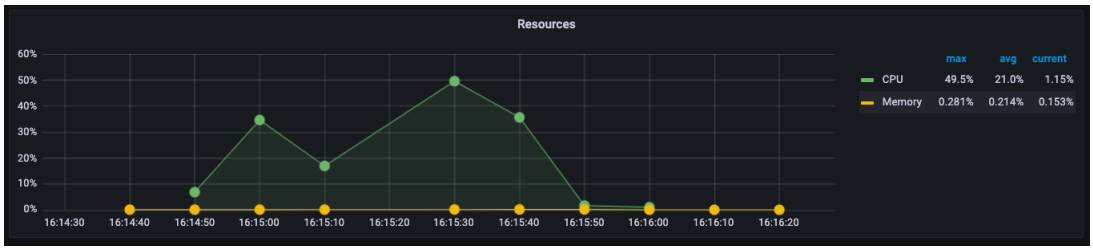
Dado que el escenario es el mismo, el estado de las requests y la cantidad de requests mandadas van a ser iguales con el nodo unico.

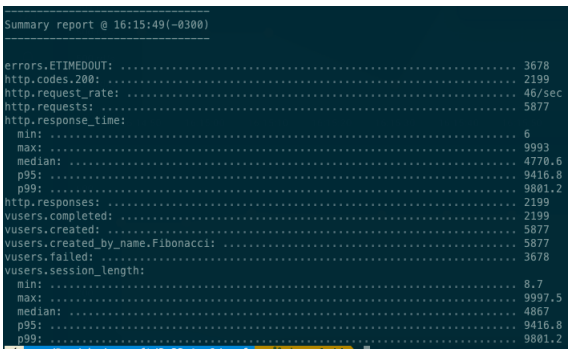


En el siguiente grafico se puede ver que, al balancear la carga sobre los distintos nodos, baja bastante el uso medio del CPU y tambien los picos maximos. Para el tiempo de respuesta, podemos notar que el pico maximo, que se alcanza al llegar al pico de requests, aumenta con respecto al nodo unico, luego se puede notar que el promedio disminuye.









Para este caso podemos observar que muchas requests terminan en timeout. Esto es entendible dado la latencia generada por nginx, la cantidad de usuarios y el **n** que usamos para la prueba

2.4. Fibonacci intensivo n=33

Para este escenario, se utiliza el endpoint que calcula el 33vo numero de la secuencia de fibonacci. Tenemos las siguientes fases:

- **1 requests/s**
- **Up to 45**
- **Keep 45 requests/s**
- **End**

2.4.1. Nodo

En el siguiente grafico podemos ver la cantidad y el estado de las requests. Vemos que al principio varias requests se completan pero luego de unos momentos todas comienzan a quedar en pending debido al timeout

Se observa que al aumentar el n a 32 el uso del CPU aumenta drásticamente, llegando incluso a ocupar el 10 % de la CPU. Esto tiene sentido dado que el algoritmo tiene una complejidad temporal de 2^N, pasa de tener que hacer 32 operaciones por request a 8,5 mil millones de operaciones por request.

2.4.2. Nodo replicado

La diferencia que podemos observar en el caso del nodo replicado es que se lograron completar más requests antes de que comiencen a generarse los timeouts

En este gráfico es donde mas se puede apreciar la ventaja de tener un load balancer con nodos replicados, ya que el tiempo de respuesta disminuye bastante en comparación a un único nodo y también el uso de CPU. El tiempo de respuesta se reduce en un poco mas de 1 segundo en promedio y el consumo de CPU disminuye significativamente, pasa de tener un avg de 97 % a 81 %. En comparacion al PING, el tiempo que tardaba el load balancer en asignar la carga a los distintos servers, era mayor a lo que tardaba la request en si, pero en Fibonacci, lo que tarda el load balancer es despreciable frente a la cantidad de operaciones que debe hacer este endpoint.

2.5. Analisis de Bbox-1

Examinamos ambos endpoints del bbox utilizando un escenario similar a cuando analizamos el endpoint *Ping* a un rate de requests constantes.

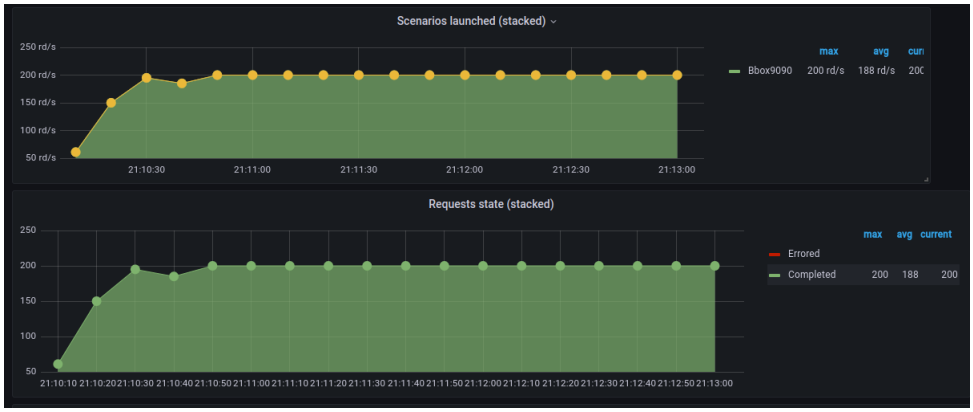


Figura 8: Gráfico requests bbox-1

Podemos observar que el response time de las requests aumenta hasta un pico en el primer momento en el que la cantidad de requests también llega un pico, y luego cuando estas últimas se estabilizan, el response time baja y permanece constante.
También se observa que el uso de la cpu es mínimo por lo que se asume que las operaciones no requieren gran cantidad de cálculos.

Figura 9: Gráfico response bbox-1

2.6. Analisis de Bbox-2

Se observa que sometiendo el endpoint a los mismos escenarios, se comportan de una forma similar.

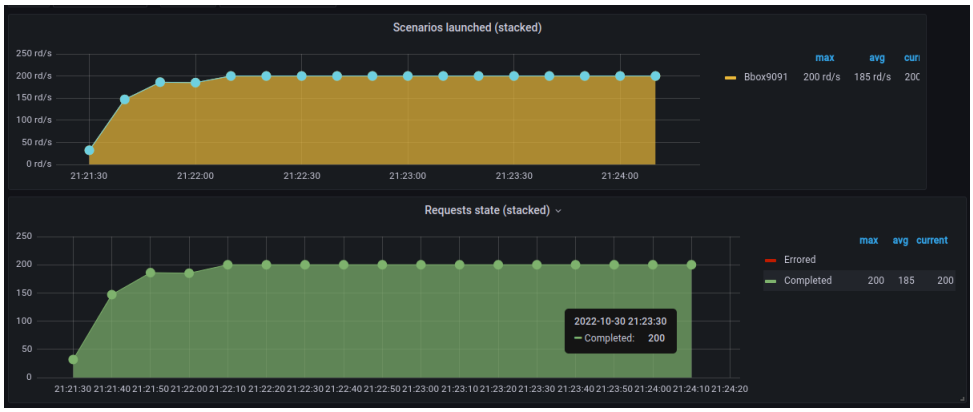


Figura 10: Gráfico requests bbox-2

Por otro parte, se entiende a tan pocas requests por segundo el asincrónico y sincrónico no difieren en su performance ni en su disponibilidad. Para tener una clara diferencia se debe hacer un análisis más exhaustivo que se realizará en la sección siguiente.

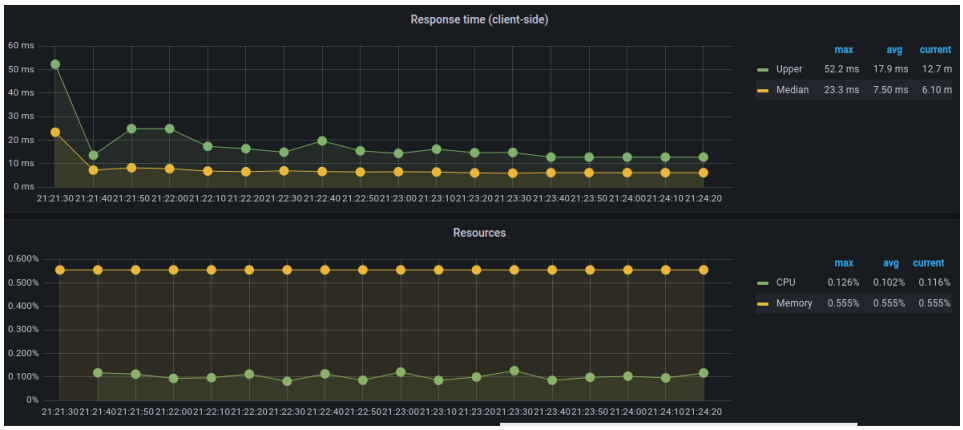


Figura 11: Gráfico responses bbox-2

3. Sección 2

En esta sección determinaremos cuál de las bbox corresponde a un servicio sincrónico y cuál asincrónico. Así como definiremos para el caso sincrónico su cantidad de workers y la demora en responder para ambos servicios.

3.1. Sincrónico - Asincrónico

En el caso **asincrónico** el sistema no llega a colapsarse como en el caso sincrónico, a pesar de ser sometido a un flujo de requests similar. Incluso con grandes cargas logra seguir respondiendo. Esto se debe a que en el caso asincrónico las requests se responden cuando sea conveniente (o posible) y no necesariamente inmediatamente al ser recibidas, como sí sucede en el caso sincrónico.

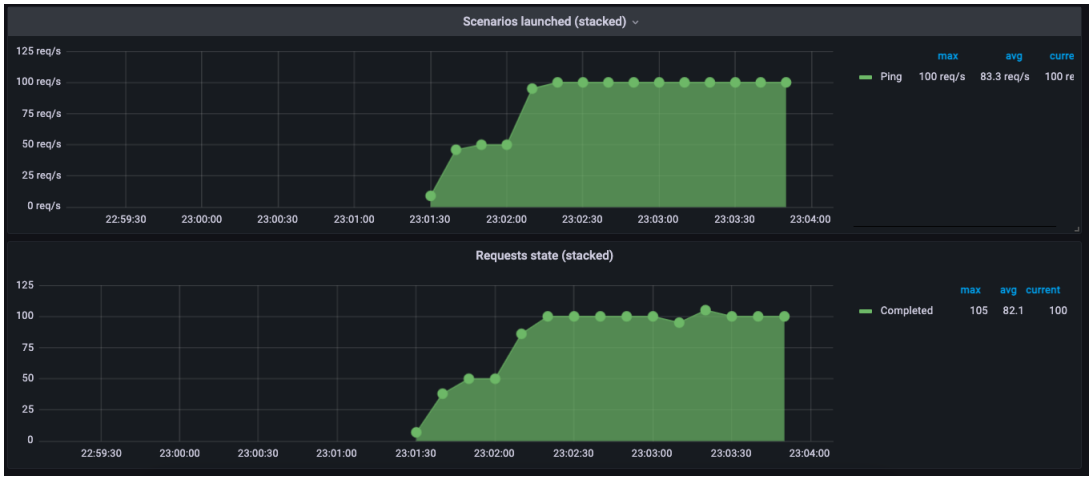


Figura 12: Gráfico response status sync bbox-2

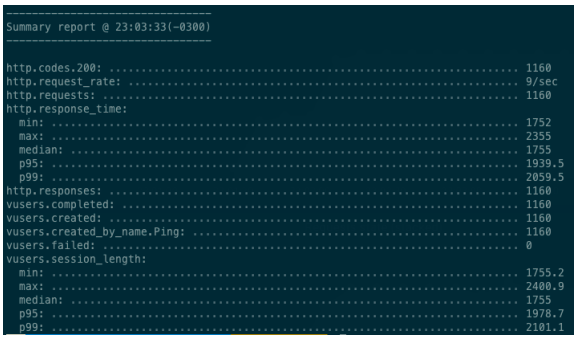


Figura 13: Summary async bbox-2

En el caso **sincrónico** podemos observar como a partir de una cantidad determinada de requests, el sistema se satura. Dado que la manera de resolverlas consiste en recibir las requests y procesarlas inmediatamente para dar una respuesta. Cuando el caudal de requests aumenta, sucederá que eventualmente en un lapso de tiempo, el sistema no podrá seguir respondiendo puesto que colapsará.

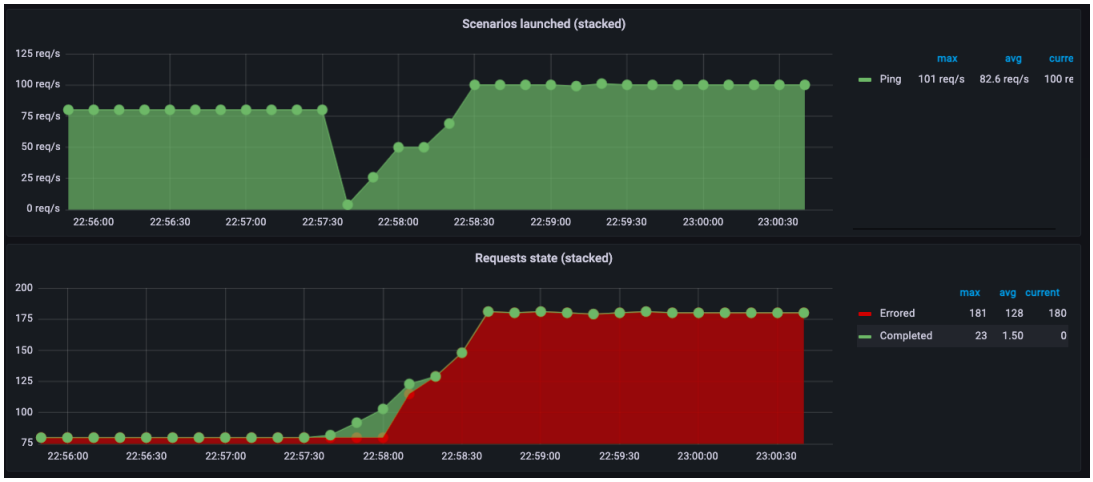


Figura 14: Gráfico response status sync bbox-1

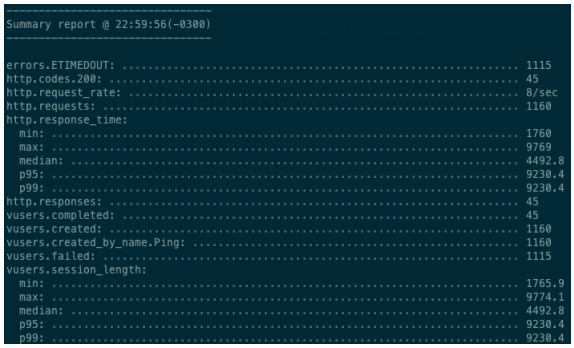
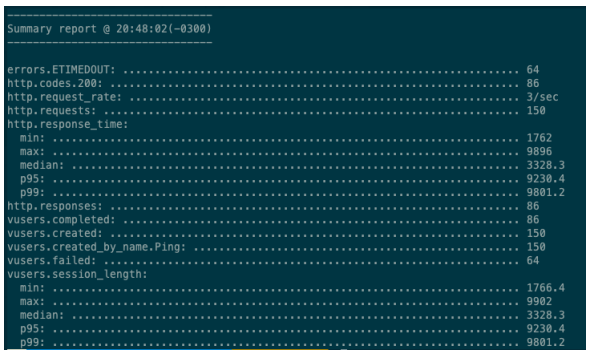


Figura 15: Summary sync bbox-1

3.2. Cantidad de workers

Para medir la cantidad de workers corrimos un escenario que consiste ir aumentado el request rate hasta alcanzar el pico máximo de requests por segundo que soporta el servicio sincrónico. De esta manera obtenemos el máximo rate de requests por segundo que soporta y luego obteniendo el rate de requests por segundo de un worker, podemos hacer una estimación de la cantidad de workers con los que cuenta dicho servicio. Para obtener el rate de requests por segundo de cada worker, tenemos el tiempo que tarda una request en ser procesada que es de 1.76 segundos y por lo tanto el rate de cada worker sera $1/1{,}76 \approx 0{,}556$.

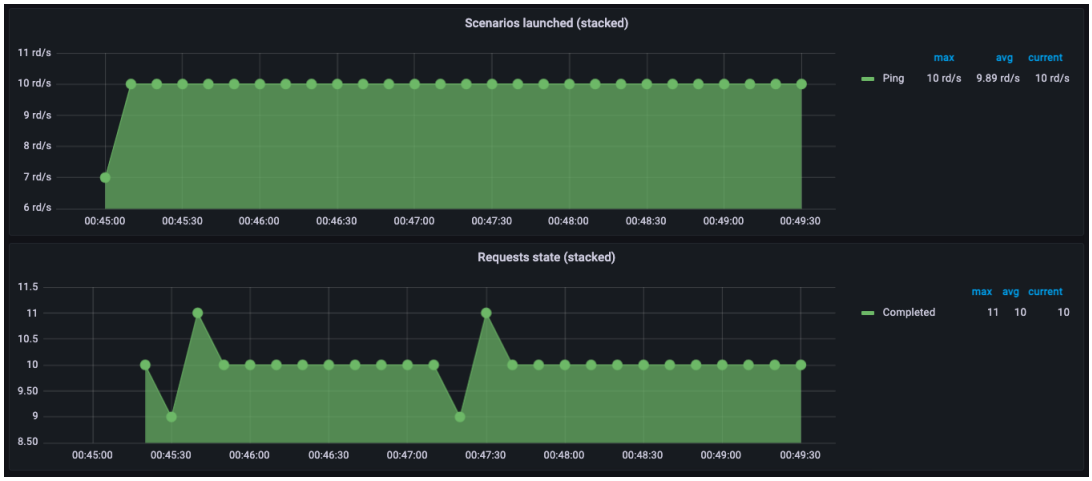


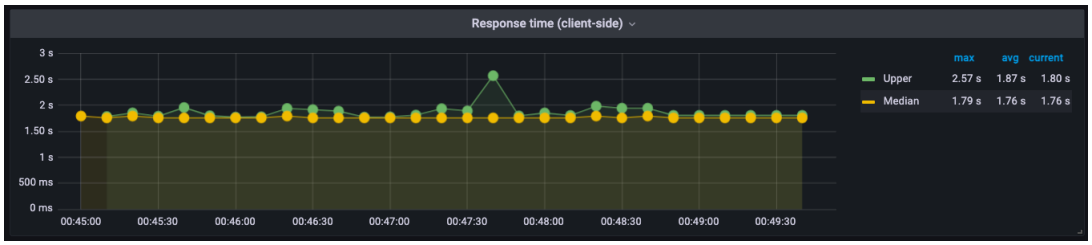
Viendo este summary podemos saber que ya con un rate de 3 requests por segundo, empiezan a aparecer requests que son rechazadas, por lo tanto en funcion del escenario que creamos, sabemos que el rate es de 2 req por segundo y por lo tanto concluimos que la cantidad de workers es de $2/0{,}556 \approx 4$

3.3. Response Time

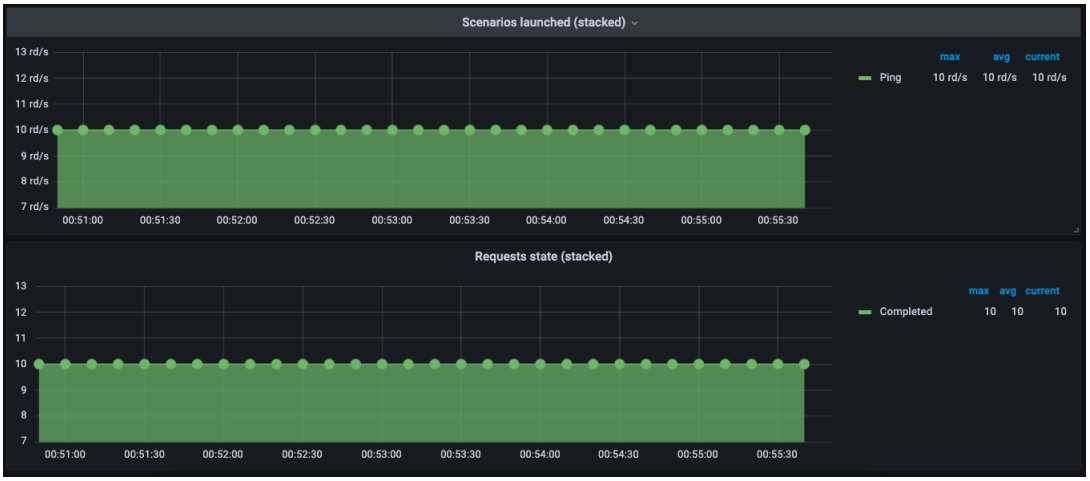
Para medir el reponse time de ambos servicios, los someteremos a un escenario en el que recibiran un flujo liviano de requests, de manera que ambos servicios puedan responder y no haya requests en pendientes o fallidas.

3.3.1. Response Time Bbox 1





3.3.2. Response Time Bbox 2



Mirando ambos gráficos podemos ver que el tiempo de respuesta de dichos servicios se encuentra en el rango de 1.8 a 1.9 segundos y que los tiempos promedios de ambos son prácticamente iguales.

Por lo tanto podemos concluir que ambos servicios tienen el mismo tiempo de respuesta.

4. Sección 3

4.1. Hipótesis

Antes de comenzar este experimento estableceremos algunas hipótesis sobre las dimensiones del problema. Teniendo en cuenta que en FIUBA hay aproximadamente 8500 alumnos y 120 prioridades distintas, alrededor de 70 alumnos nuevos son habilitados para realizar sus inscripciones cada 30 minutos (duración de cada turno). Considerando que, en promedio, cada alumno se inscribe en 4 materias, la sesión de inscripción de un alumnos constaría de:

- 1 request para inicio de sesión
- 1 request para seleccionar la carrera

- 12 requests para inscribirse a sus materias (4 materias* (materias inscripto + ver materias disponibles + inscripción materia))
- 1 request para cerrar sesión

Aproximadamente serían 15 requests por sesión de inscripción por cada alumno. Por lo tanto, en cada franja de inscripción habría aproximadamente 1050 requests, es decir 2100 req/hora, suponiendo que todos los alumnos de cierta prioridad se anotan en la franja de tiempo correspondiente a la misma.

4.2. Modelado de endpoints

En esta sección comentaremos un poco el comportamiento de cada endpoint y estimaremos su tiempo de respuesta. En la mayoría de ellos, para poder aproximar el tiempo de demora tomamos como referencia dicho valor para requests reales realizadas en el SIU guarani obtenidas mediante la consola de Google Chrome.

- Iniciar sesión:

  A través de un POST debería validar las credenciales ingresadas por el usuario. Tiempo de demora promedio: 250 ms

- Seleccionar carrera : A través de GET deberia cambiarte a la carrera seleccionada. Tiempo de demora promedio: 100 ms

- Mis Inscripciones: A través de un GET debería devolver la lista de materias en las que el alumno esta inscripto. Tiempo de demora promedio: 250 ms.

- Materias disponibles: A través de un GET debería devolver la lista de materias disponibles a las que un alumno podría inscribirse. Tiempo de demora promedio: 800 ms.

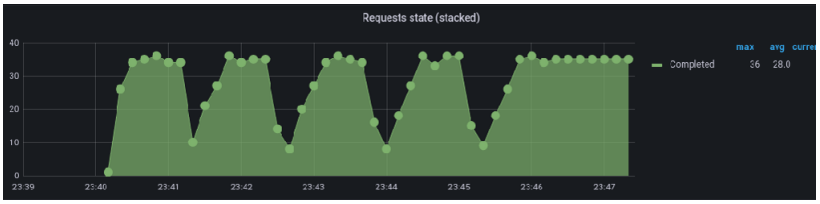
- Inscribirse: A través de un POST debería inscribir al usuario a la materia seleccionada. Tiempo de demora promedio: 350 ms

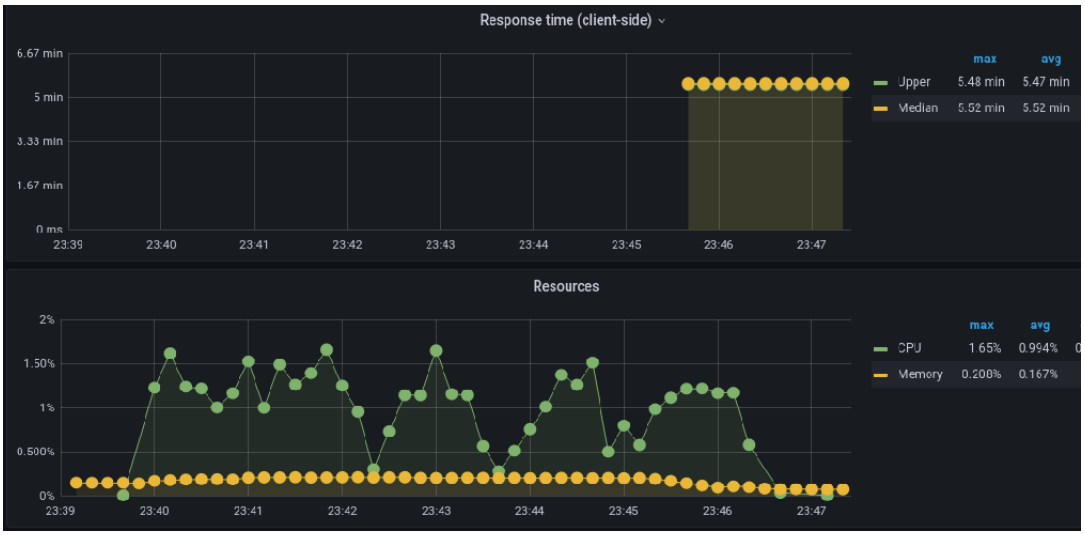
- Cerrar sesión: A través de un GET debería devolver la pagina de inicio del SIU GUARANI. Tiempo de demora promedio: 150 ms

4.3. Escenarios planteados

4.3.1. Escenario numero 1

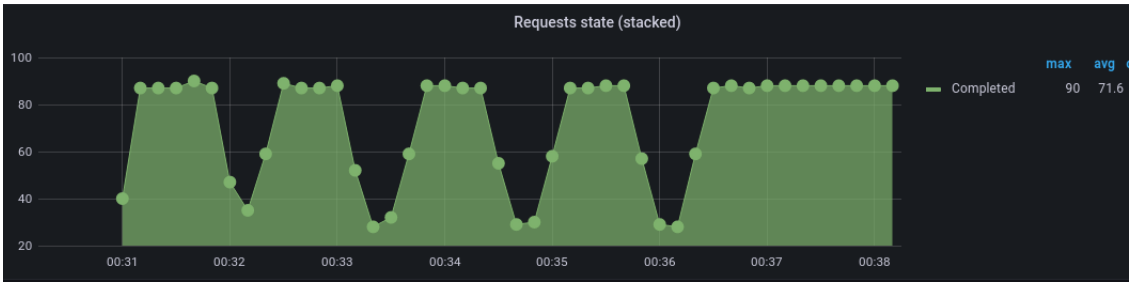
En este escenario planteamos el caso en el cual todos los alumnos correspondientes a su franja horaria deciden anotarse en sus materias ni bien se ses habilita la posibilidad, es decir, en el primer minuto.

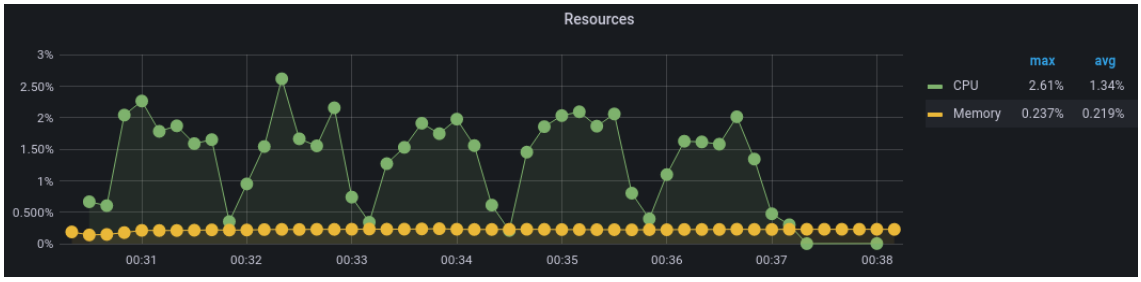




4.3.2. Escenario numero 2

En este escenario tenemos la siguiente situación, varios alumnos no se inscribieron en su franja correspondiente por lo que al abrirse la siguiente tenemos más alumnos anotándose en una misma franja horaria, generando un alto trafico de datos en el sistema.

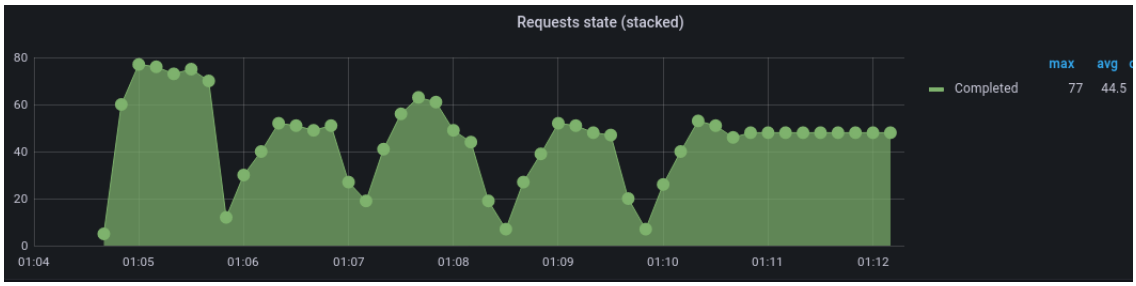


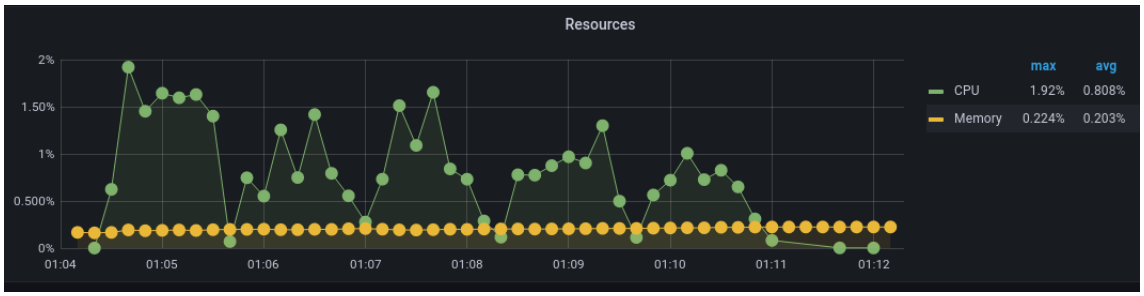


4.3.3. Escenario numero 3

Para este escenario tenemos una situación mas real, vamos a tener en el sistema 175 usuarios de los cuales no todos van a inscribirse sino que existe un 40 por ciento de probabilidad de que sean de una franja horaria posterior a la del momento por lo que entran al sistema a revisar la oferta académica.

En el summary report podemos ver que 97 alumnos fueron los que se inscribieron y 78 solo ingresaron a revisar la oferta académica






```
All VUs finished. Total time: 6 minutes, 29 seconds

------------------------------
Summary report @ 01:10:56(-0300)
------------------------------

http.codes.200: ..... 1689
http.request_rate: ..... 4/sec
http.requests: ..... 1689
http.response_time:
  min: ..... 101
  max: ..... 841
  median: ..... 354.3
  p95: ..... 804.5
  p99: ..... 820.7
http.responses: ..... 1689
vusers.completed: ..... 175
vusers.created: ..... 175
vusers.created_by_name.inscripcion: ..... 97
vusers.created_by_name.revisionOferta: ..... 78
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 127213.4
  max: ..... 328281.6
  median: ..... 331165.6
  p95: ..... 331165.6
  p99: ..... 331165.6
```

4.4. Conclusiones

A pesar de las diferencias en los valores mostrados en los gráficos de recursos(), todos los escenarios reflejan un comportamiento similar con respecto al consumo de CPU, picos altos de consumo por una alta cantidad de requests y luego valles que se producen debido a que los alumnos se detienen a analizar las materias disponibles o observar sus inscripciones por ejemplo.

Desde nuestra perspectiva las métricas obtenidas reflejan bastante bien el funcionamiento de un sistema de inscripciones. Este, debería estar preparado para soportar ráfagas de grandes cantidades de requests en poco tiempo y tener en cuenta que no solo aquellos alumnos que pueden inscribirse en ese momento van a utilizar el sistema. Anteriormente,este era el problema principal del SIU, muchos alumnos cuyo turno para inscribirse no había llegado,ingresaban al sistema para revisar la oferta académica. Esto hacía que el número de usuarios en el sistema fuese 3 o 4 veces mayor al número de alumnos asignados por prioridad y el sistema colapsara. Justamente, se le pedía a los alumnos que solo ingresaran al sistema cuando llegara su turno de anotarse.

En conclusión, el sistema de inscripciones debería estar preparado ante estos picos de requests altos, probablemente debería contar con un balanceador de carga y varios nodos replicados para poder distribuir las requests y no colapsar un único nodo concentrando todo el procesamiento en él.