Arquitectura de Software (75.73)

Trabajo Práctico Nº 1 (Re-entrega)

Grupo: Quality Crunchers

Nombre	Padrón
Cozza, Fabrizio	97.402
Prieto, Pablo	91.561
Condori, Guillermo	98.688

Escenarios

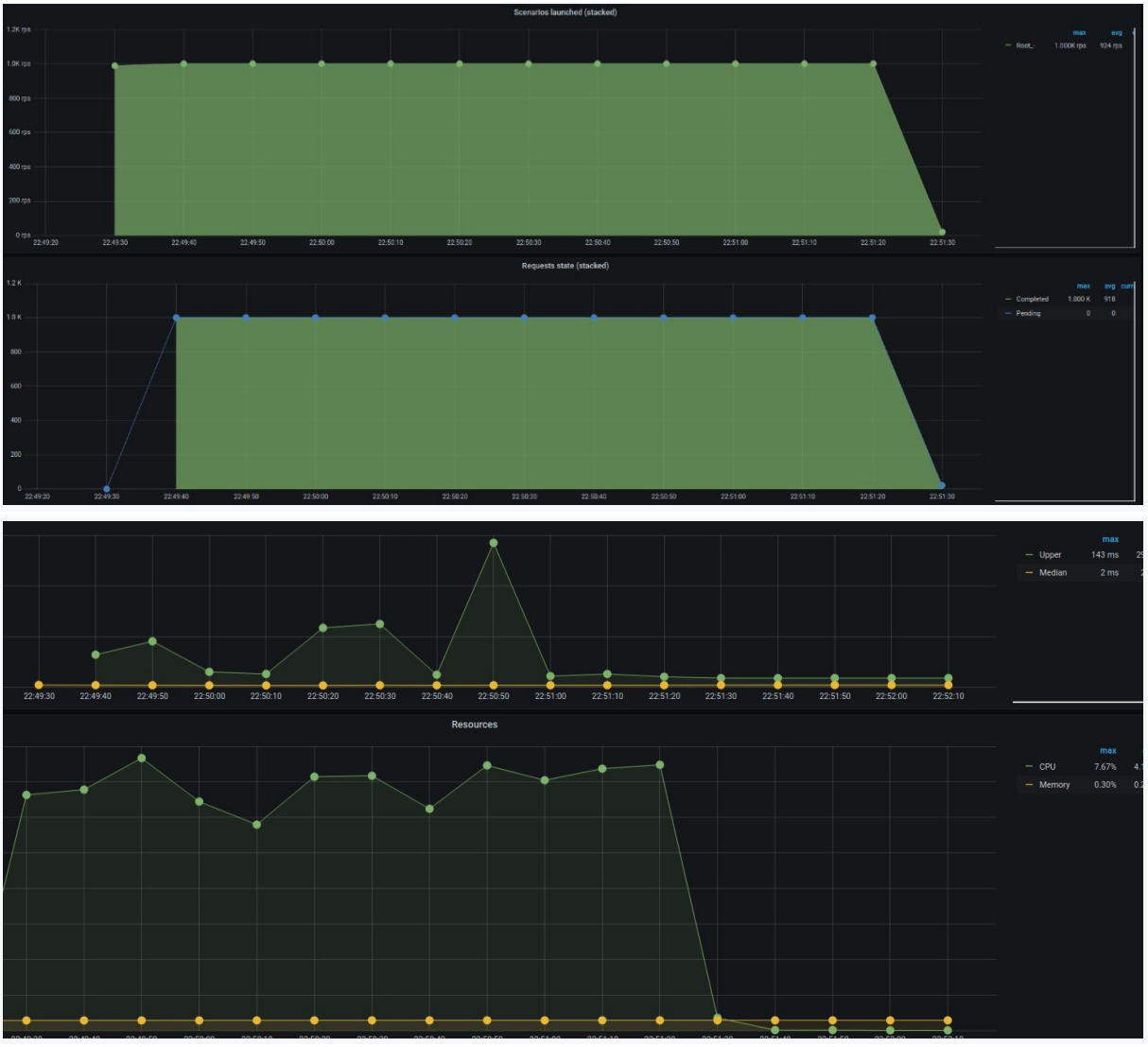
Sección 1

Caso 1: Ping Endpoint

Lo que primero se intentó ver en este caso fue el límite del servidor al enviar reiteradamente una gran cantidad de solicitudes para este endpoint que prácticamente no necesita procesamiento.

- **Server**: Servidor en python con gunicorn, usando un solo worker sincrónico (el del tipo default). Un solo container.

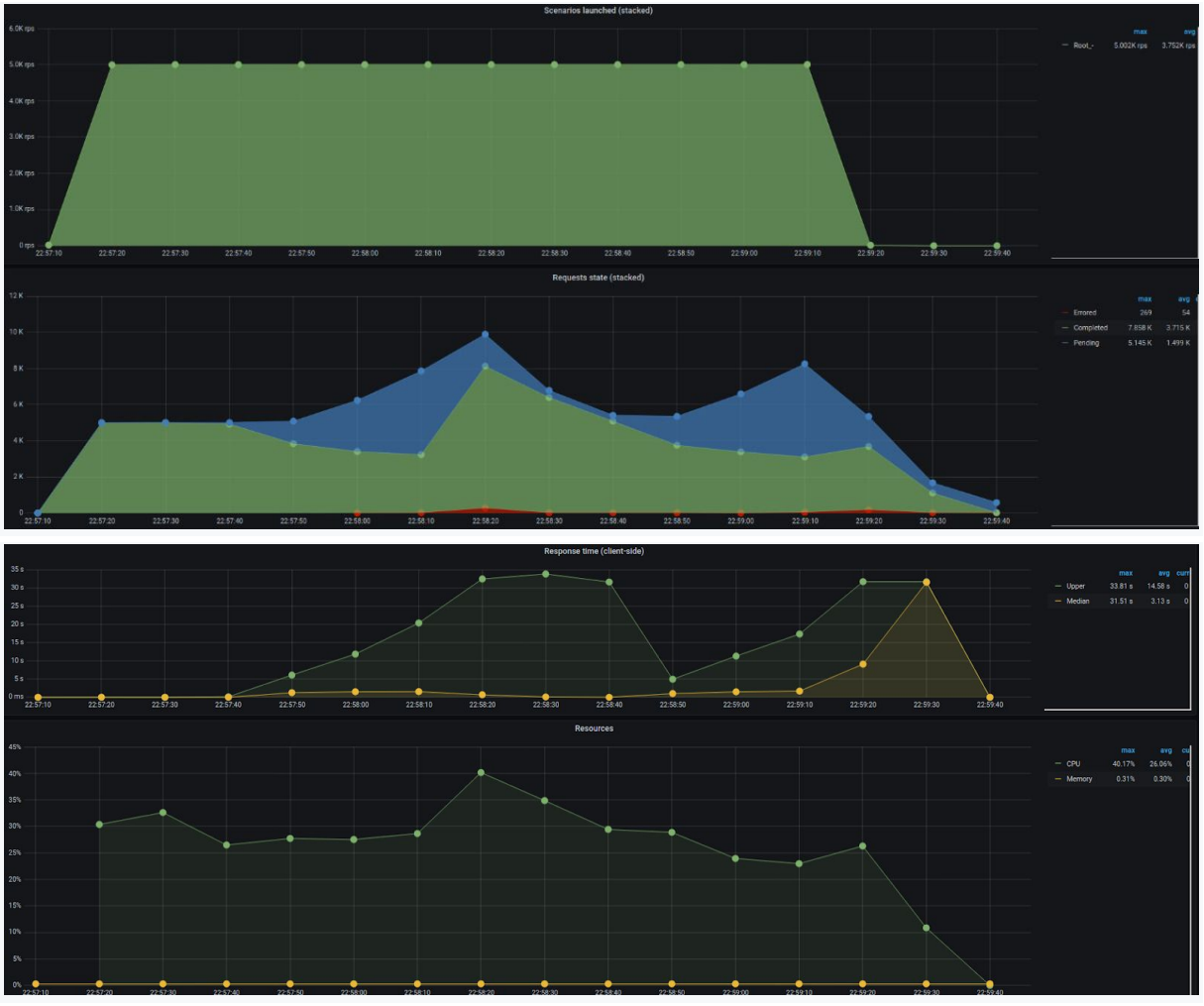
100 requests:



Summary report:

Scenarios launched: 12010
Scenarios completed: 12010
Requests completed: 12010
Mean response/sec: 92.01
Response time (msec):
    min: 0.8
    max: 141.5
    median: 1.1
    p95: 1.5
    p99: 5.6
Scenario counts:
    Root (/): 12010 (100%)
Codes:
    200: 12010

500 requests:



```
Summary report:
  Scenarios launched:  60010
  Scenarios completed: 59446
  Requests completed:  59446
  Mean response/sec: 385.27
  Response time (msec):
    min: 0.7
    max: 33807
    median: 514
    p95: 16694.8
    p99: 31659.6
  Scenario counts:
    Root (/): 60010 (100%)
  Codes:
    200: 59446
  Errors:
    ECONNRESET: 546
    ESOCKETTIMEDOUT: 18
```

Comentarios de la re-entrega

Corrección 1: Gunicorn no está paralelizando. El endpoint tarda unos pocos milisegundos y toma el siguiente request.

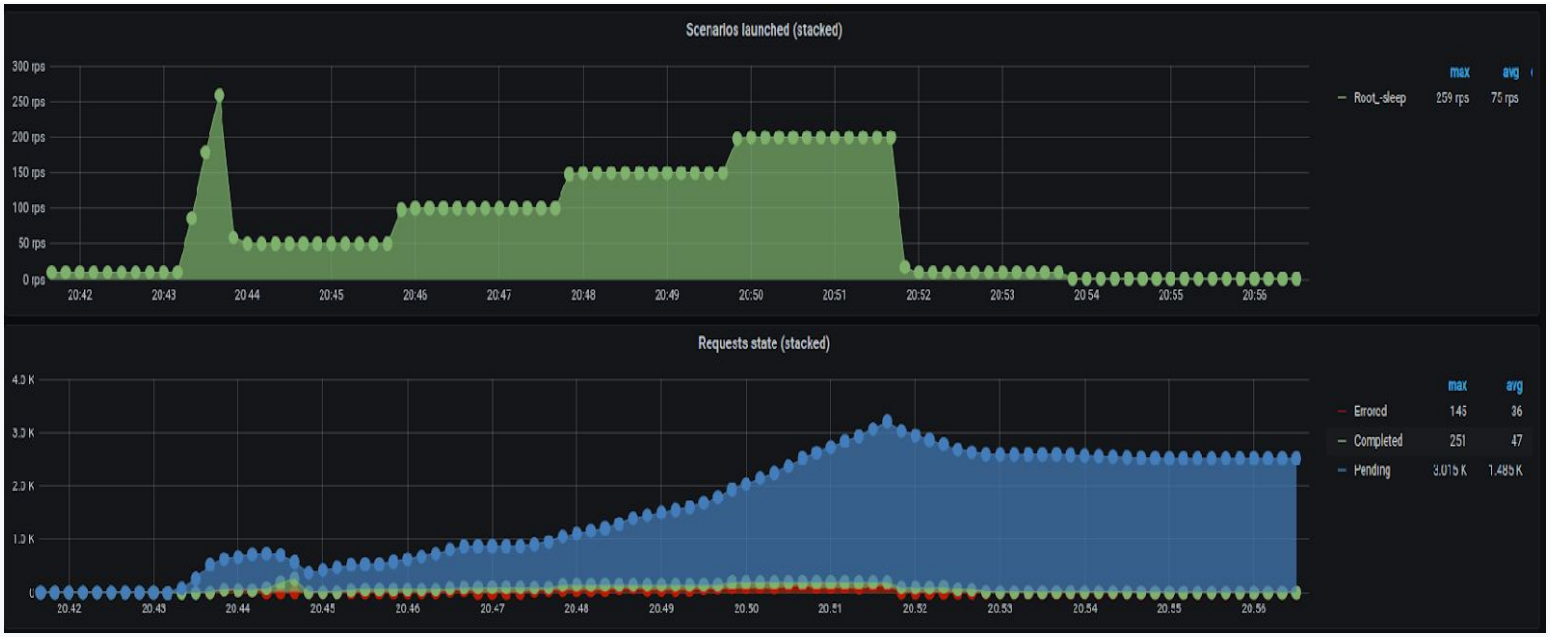
Podemos ver, para el caso de 100 request, que en realidad el servidor no está realizando las solicitudes en paralelo, sino que el servidor de gunicorn con un solo worker logra procesar todos los request, ya que tarda unos pocos milisegundos (1,1 msec de mediana) por cada uno, debido probablemente a la potencia del procesador utilizado (i7-7700k).

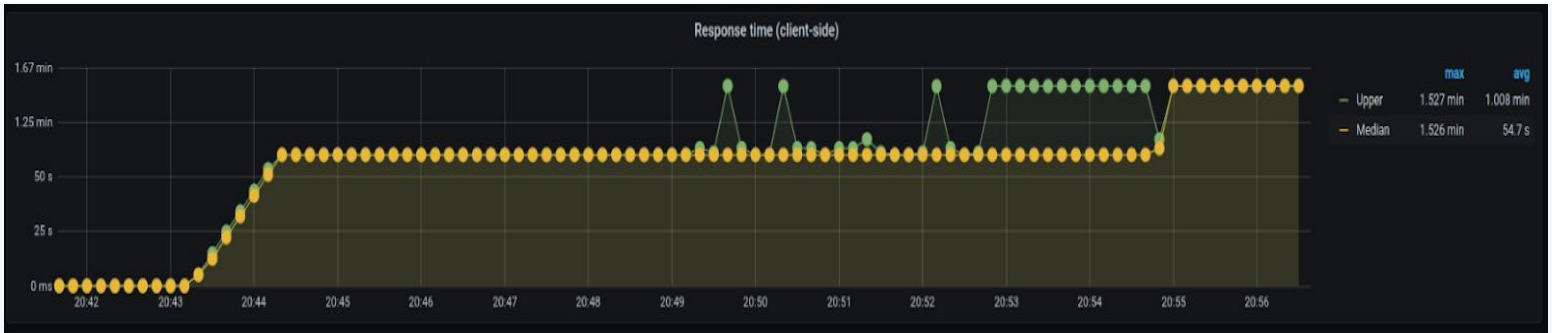
Recién en el caso de 500 request podemos ver que el servidor se estresa generando tiempos de respuesta mayores (30 segundos) debido a la carga impuesta.

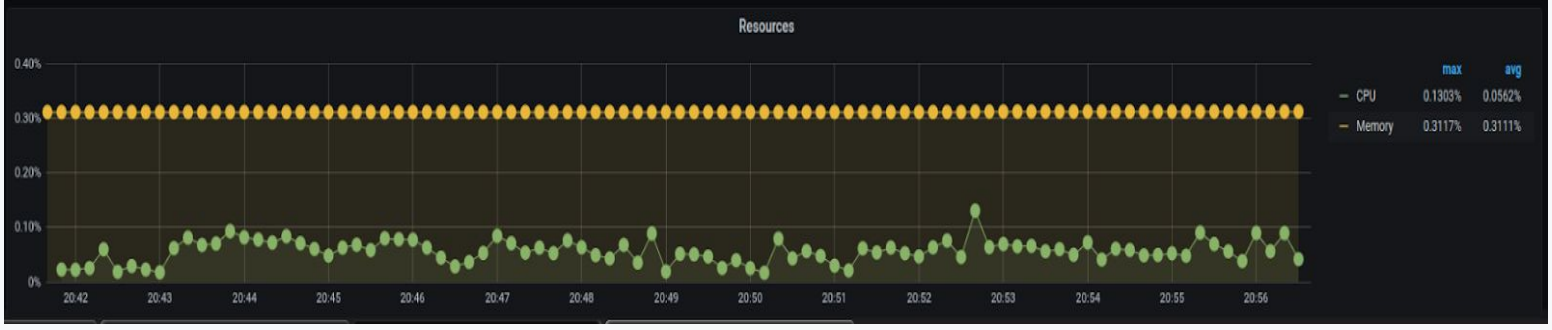
Caso 2: Proxy/timeout Endpoint

Aquí se intentó ver qué causaban solicitudes en los que su objetivo es dormir el proceso 5 segundos, se los fue incrementando progresivamente para notar su efecto.

- **Server**: Servidor en python con gunicorn, usando un solo worker sincrónico (el del tipo default). Un solo container.







```
Summary report:
  Scenarios launched:  6655
  Scenarios completed: 4145
  Requests completed:  4145
  Mean response/sec: 9.29
  Response time (msec):
    min: 5015.3
    max: 91623.8
    median: 60002.8
    p95: 60006.8
    p99: 67218.1
  Scenario counts:
    Root (/sleep): 6655 (100%)
  Codes:
    200: 12
    504: 4133
  Errors:
    ECONNRESET: 2510
```

Comentarios de la re-entrega

Corrección 2: Tiene que haber algún error en la forma en la que lanzaron Gunicorn o en la implementación de la app. Con 1 worker sincrónico, no paraleliza los requests. Además, aún cuando lo hiciera, la demora del endpoint sleep es de 5 segundos, no puede tener tiempos de respuesta de 8 milisegundos (gráfico pág. 12).

El error de los tests sucedía por un carácter '/' erróneo al final de la URL de sleep en los tests correspondientes de artillery. Donde para los casos de node sleep funciona con la '/' al final (/sleep/) y para gunicorn no funciona arrojando errores 404.

Ahora que arreglamos los casos podemos ver el comportamiento esperado en gunicorn, a medida que llegan nuevas requests, las mismas deben esperar para ser completadas, generando una larga cola de solicitudes pendientes y errores, debido al comportamiento sincrónico del mismo.