Trabajo Práctico 1

Reporte de
Pruebas de Carga

75.61 Taller de Programacion III

Segundo Cuatrimestre 2021 - FIUBA

Nombre y Apellido	Padrón
Franco Giordano	100608

Objetivos

Se solicita el desarrollo de un sitio web institucional con los siguientes requerimientos funcionales:

- Se debe registrar las visitas en las distintas páginas: home, jobs, about, about/legals y about/offices.
- El conteo de visitas debe ser incremental y por cada página de forma individual.
- En cada sección se debe visualizar el contador de sus visitas totales recibidas.

Como requerimientos no funcionales, tenemos los siguientes:

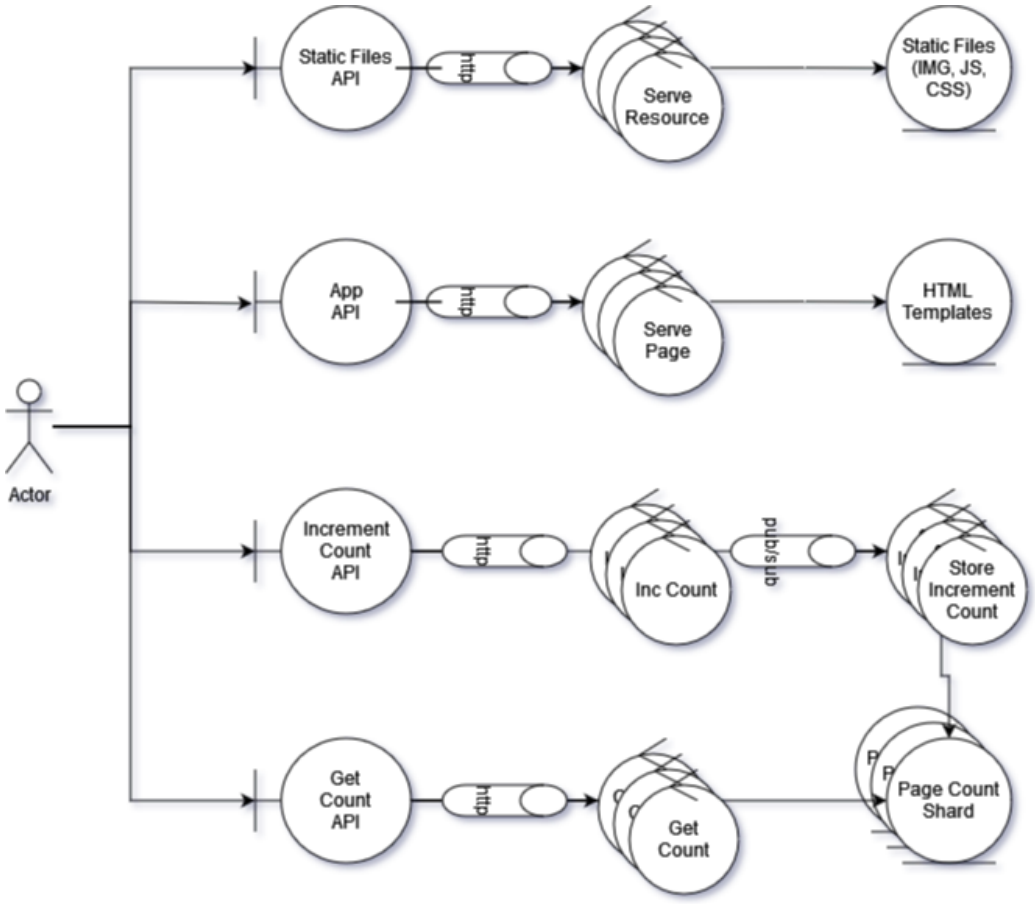
- El sistema debe escalar automáticamente con el tráfico recibido.
- El sistema debe mostrar alta disponibilidad hacia los clientes.
- El sistema debe ser tolerante a fallos como la caída de procesos.

El presente reporte documenta las sucesivas iteraciones realizadas sobre la arquitectura propuesta. Para cada una, se parte de una o varias hipótesis y se la contrasta con una prueba de carga para verificar la veracidad de la misma.

El objetivo final resulta entonces desarrollar un sistema altamente disponible acompañado de pruebas de performance que verifiquen su comportamiento ante grandes cargas.

Iteración 0

Partimos de un sistema inicial con varios componentes ya distribuidos:



Nuestro sistema está compuesto entonces por 4 Cloud Functions:

- Serve page: sirve el HTML pedido por query param.
- Get Count: Dado un tipo de visita mediante query param, reduce los valores de los shards y lo devuelve.
- Inc Count: Dado un tipo de visita mediante query param, lo publica en un tópico de Cloud Pub/Sub.
- Store Incremented Counter: Dado un mensaje de Cloud Pub/Sub con un tipo de visita, incrementa el contador de un shard aleatorio.

La API de Static Files es administrada en su totalidad por GCP, por lo que la excluiremos del análisis.

Comenzamos además con **2 shards por página** en los contadores de visitas.

Breakpoint Test

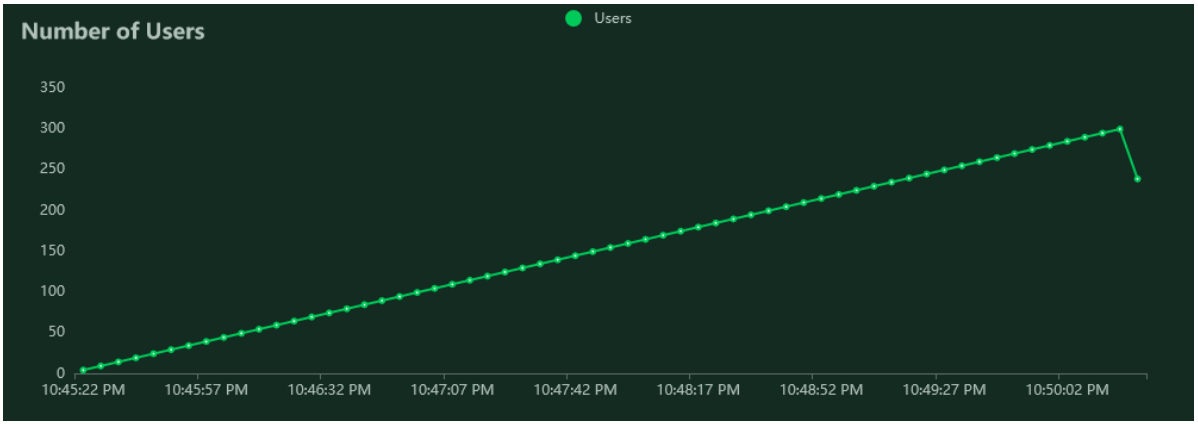
Comenzamos entonces realizando un Breakpoint test con Locust para encontrar el punto de quiebre del sistema. Trabajaremos con los siguientes parámetros:

- Cantidad máxima de usuarios: 30000 (“infinito”)
- Incrementos: +1 usuario / segundo
- Tiempo total de corrida: 7:45:18 PM - 7:50:22 PM (5 minutos)

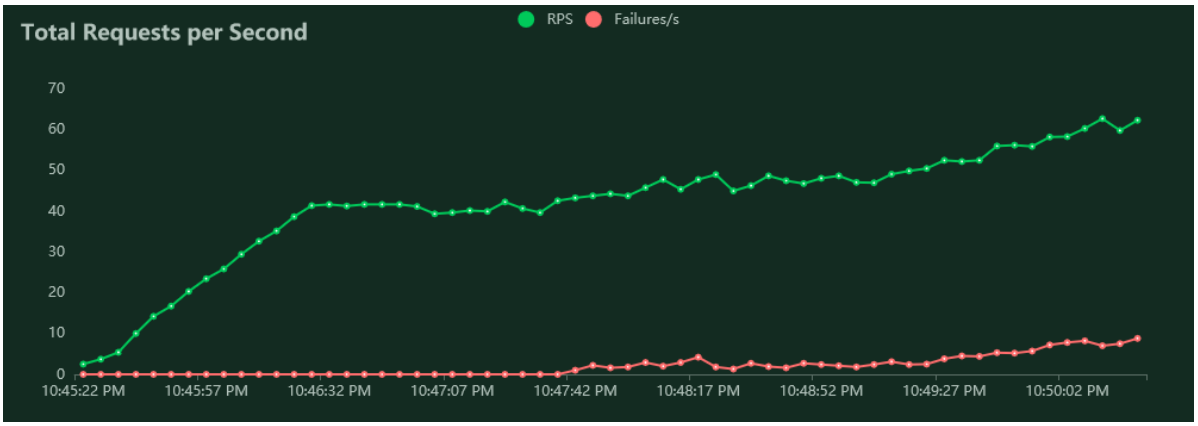
- Límite de instancias: 2 para cada Cloud Function

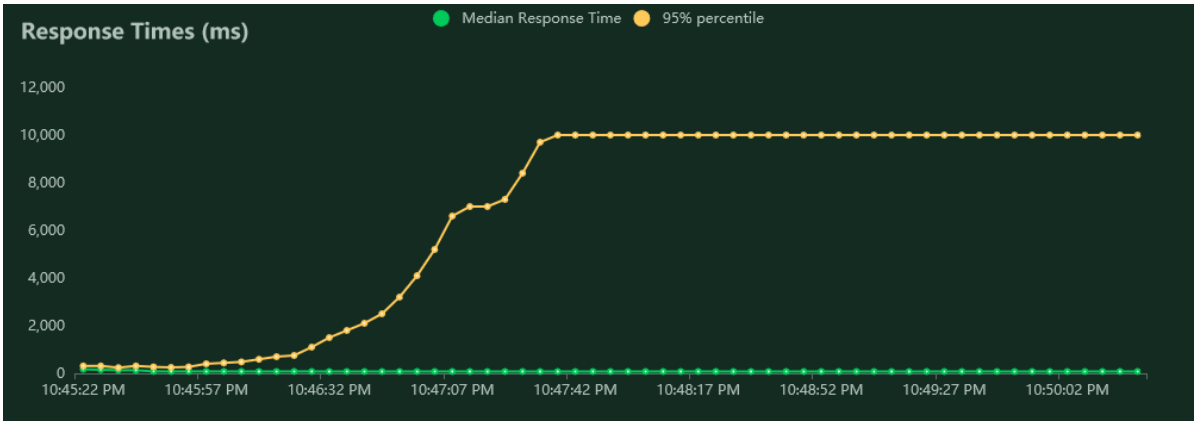
Se puede encontrar la configuración de Locust en el Anexo.

Tendremos así la siguiente curva de carga:



Viendo los resultados, luego de 1 minuto (10:46:32PM) los RPS dejan de incrementar linealmente con la carga. Pasados tan solo 2 minutos de corrida (10:47:42PM) encontramos las primeras fallas con 150 usuarios, alcanzando ~45 RPS.



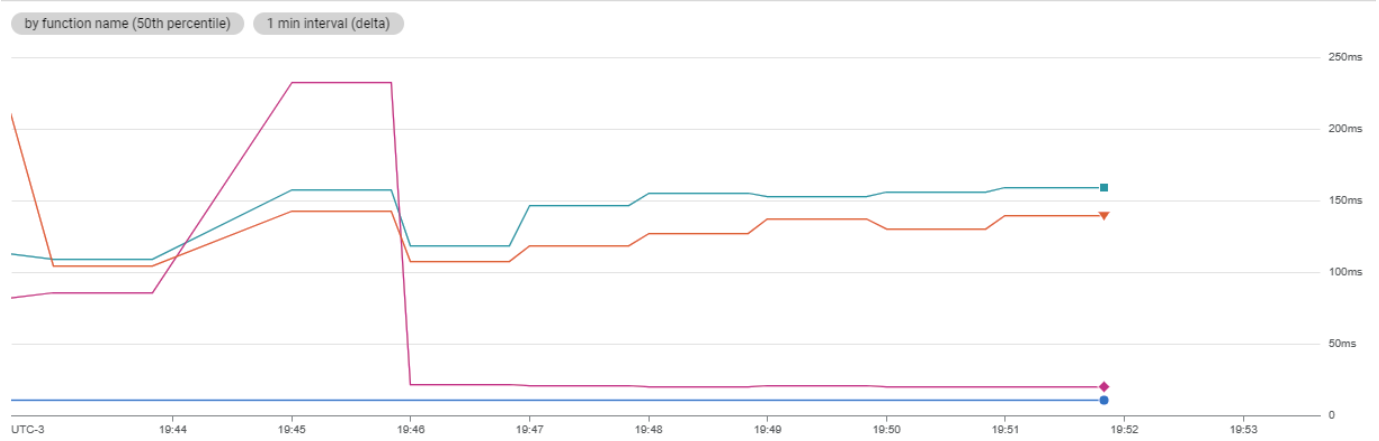


Notar las primeras fallas alrededor de 10:47:42, coincidente con el tope de 10000 ms para el 95% percentil.

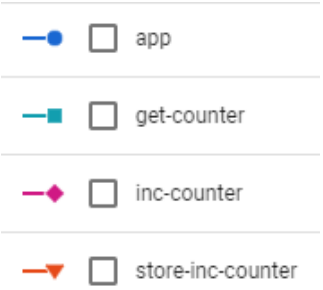
Al examinar los errores descubrimos al responsable: el único endpoint que arrojó errores al usuario durante la prueba fue `/get-counter`, con el siguiente mensaje: `429 Client Error: Too Many Requests for url: (...)/get-counter?visit_type=(...).` Estudiando los tiempos de Locust, vemos una enorme diferencia:

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)
GET	/app?view=about	63	64	66	69	81	130
GET	/app?view=about_legals	63	65	67	70	94	140
GET	/app?view=about_offices	63	64	66	69	96	140
GET	/app?view=home	63	64	65	68	86	130
GET	/app?view=jobs	63	64	66	68	79	130
GET	/get-counter?visit_type=about	9700	10000	10000	10000	10000	10000
GET	/get-counter?visit_type=about_legals	9900	10000	10000	10000	10000	10000
GET	/get-counter?visit_type=about_offices	9900	10000	10000	10000	10000	10000
GET	/get-counter?visit_type=home	9800	10000	10000	10000	10000	10000
GET	/get-counter?visit_type=jobs	9800	10000	10000	10000	10000	10000
POST	/inc-counter?visit_type=about	77	78	79	82	89	110
POST	/inc-counter?visit_type=about_legals	77	78	79	82	91	110
POST	/inc-counter?visit_type=about_offices	76	77	79	82	91	110
POST	/inc-counter?visit_type=home	77	78	80	82	90	110
POST	/inc-counter?visit_type=jobs	77	78	80	82	89	110
	Aggregated	**78**	**86**	**310**	**6000**	**10000**	**10000**

La función `/get-counter` resulta hasta dos órdenes de magnitud más lenta que sus contrapartes. Contrastando esto con las métricas de GCP, vemos los tiempos de ejecución (50% percentil):



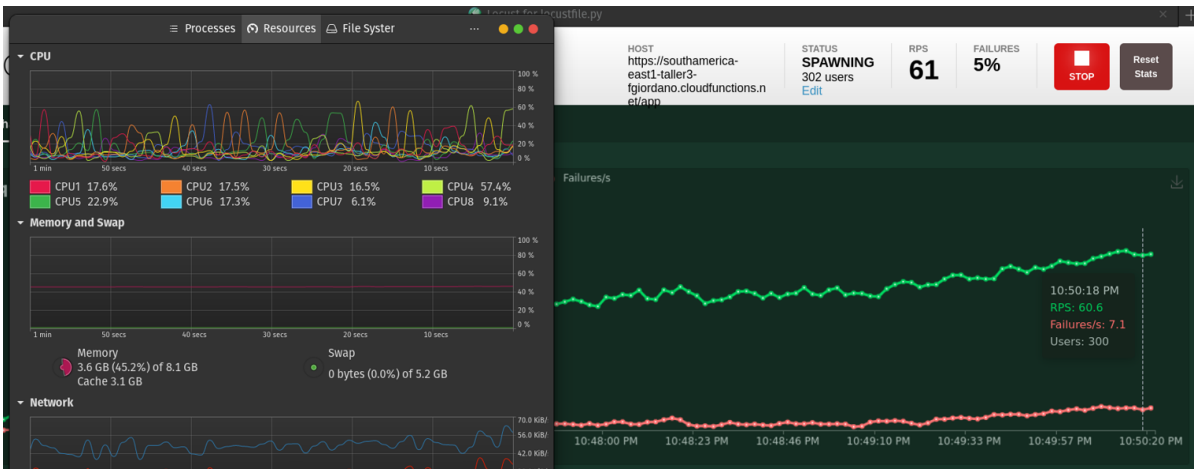
Con las siguientes referencias:



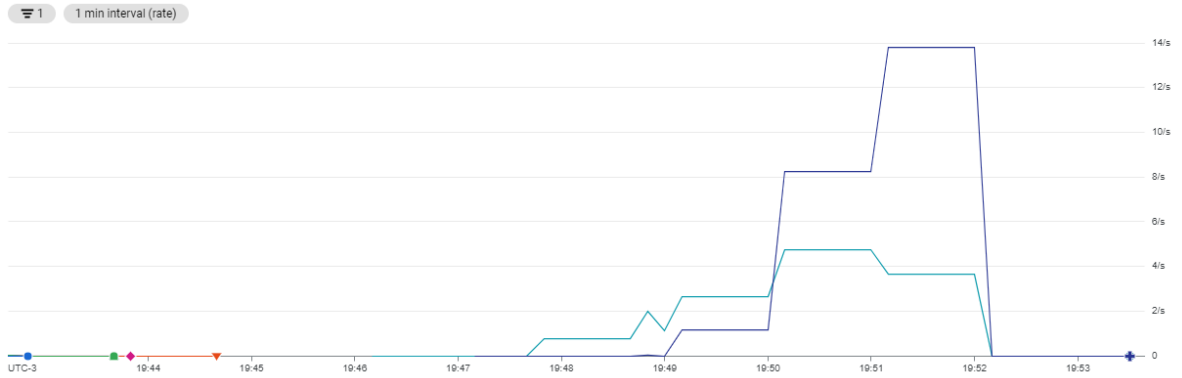
Vemos que nuevamente get-counter resulta ser la más lenta de todas las funciones, teniendo en segundo lugar al worker asincrónico store-inc-counter.

Cabe destacar la diferencia de tiempos vista en Locust (9900 ms) vs. la vista por GCP (150 ms). Atribuimos esta disparidad a un timeout interno de GCP, en particular al load balancer encargado de asignar las peticiones entrantes a las 2 instancias de get-counter. Entendemos que al no encontrar una instancia disponible luego de 10 segundos, GCP rechaza la petición con código 429 pero sin alterar las métricas de la función.

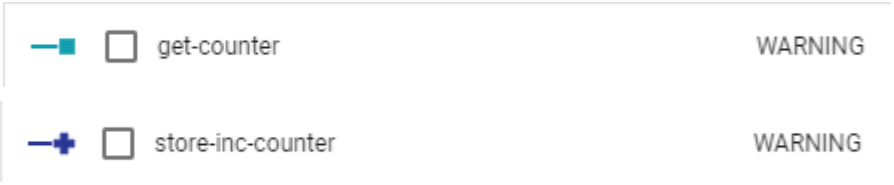
Descartamos que haya sido un problema de performance de la computadora corriendo las pruebas, pues jamás alcanzó un uso intensivo de recursos, aun en los últimos momentos del test:



Podemos, de todos modos, encontrar estas alertas en los logs, siendo cada entrada de WARNING el registro de un código 429:

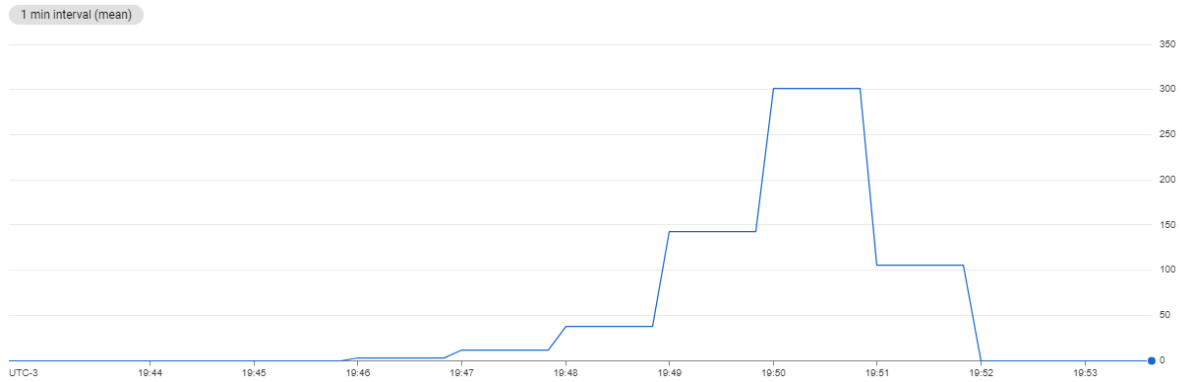


Con la referencia:



Curiosamente, vemos que store-inc-counter también presenta un cuello de botella, pero esto no es de momento un problema pues es una tarea asincrónica. Esto significa que el sistema sólo tomará más tiempo en registrar todas las visitas.

De hecho, examinando la cantidad de mensajes sin procesar en el tópico, vemos que:



Para una prueba de 5 minutos terminó de procesar todas las visitas hasta 2 minutos después, lo cual resulta aceptable.

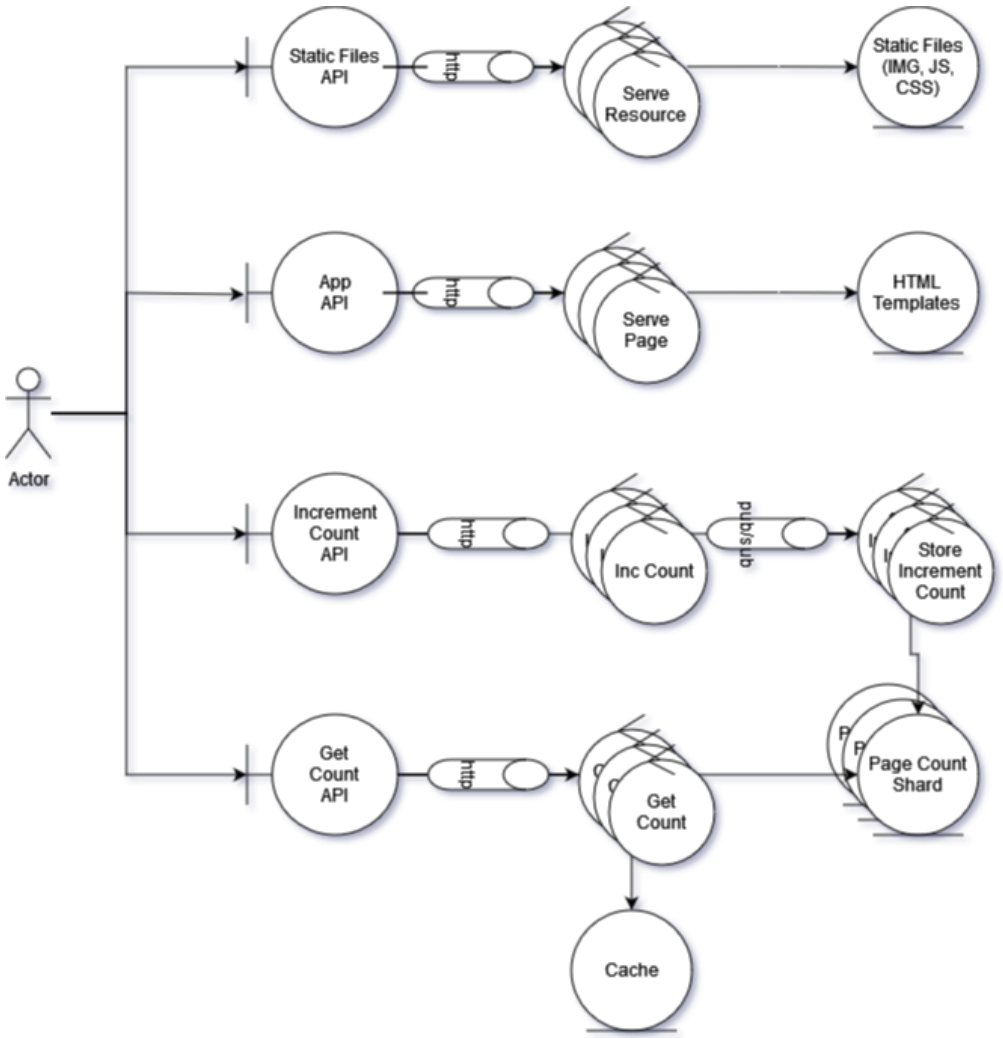
Con este análisis concluimos que get-counter resulta ser un cuello de botella, por lo que nos centraremos en resolver su performance. Manejaremos entonces la siguiente hipótesis:

“La baja performance de `get-counter` es debido a que debe obtener y reducir los contadores cada vez que es ejecutada. Podremos optimizar esto con un caché en memoria de los cómputos efectuados.”

Intentaremos poner a prueba la misma en la siguiente iteración.

Iteración 1

Tomando los resultados de la etapa anterior, proponemos una mejora a nuestra arquitectura:



La idea será ahora cachear los resultados de cada tipo con cierto timeout. Para ello, se optó por mantener el caché como una variable global del proceso para una más sencilla implementación.

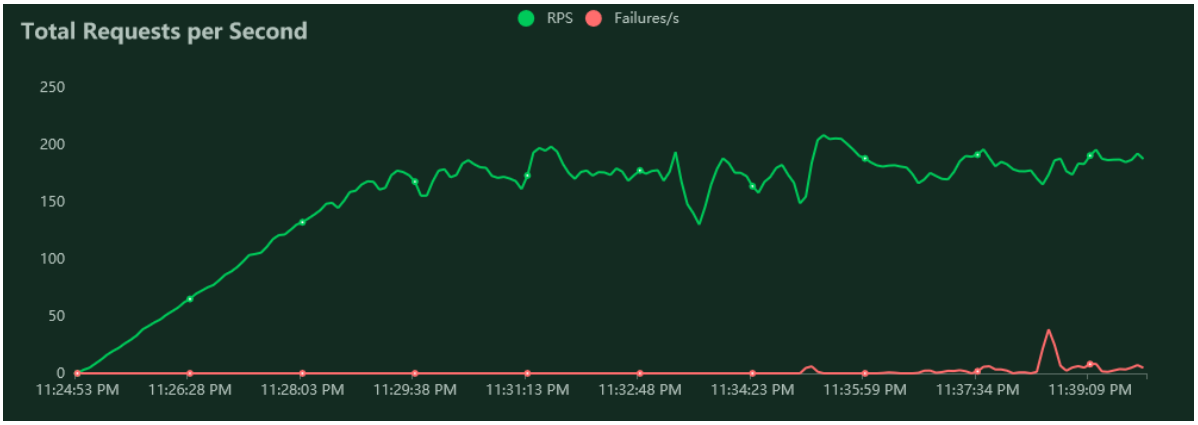
Seguimos trabajando con **2 shards por página** en los contadores de visitas, y un **timeout de caché de 10 segundos**.

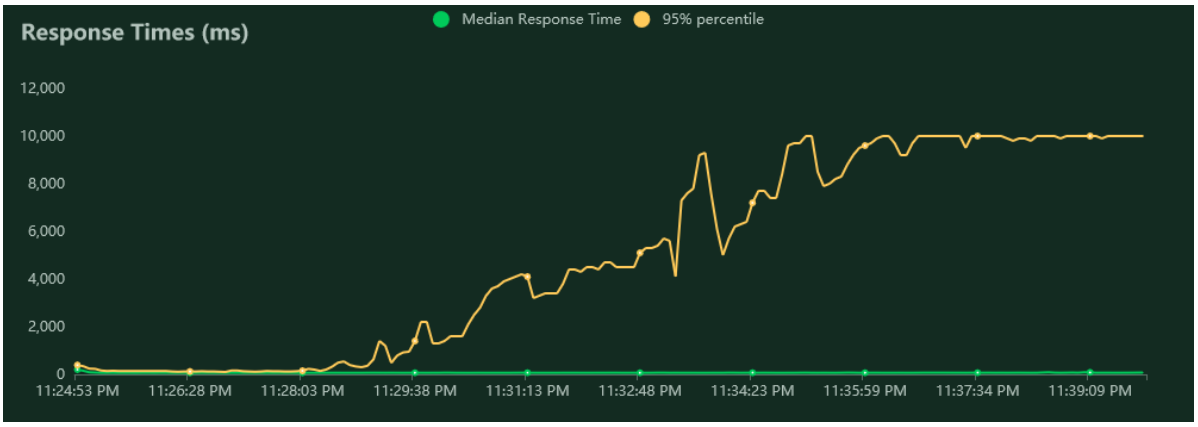
Breakpoint Test

Nuevamente, realizaremos un breakpoint test para corroborar si cambio el punto de quiebre del sistema. Trabajaremos con los mismos parámetros:

- Cantidad máxima de usuarios: 30000 (“infinito”)
- Incrementos: +1 usuario / segundo
- Tiempo total de corrida: 8:24:51 PM - 8:39:57 PM (15 minutos)
- Límite de instancias: 2 para cada Cloud Function

Tendremos así la misma curva de carga que antes, pero los resultados cambian:





Los RPS dejan de responder al aumento lineal a las 11:29:38 PM con 288 usuarios, incrementando el 95% percentil. Una vez pasadas las 11:35:59PM tenemos los primeros errores alcanzando ~670 usuarios con ~170 RPS.

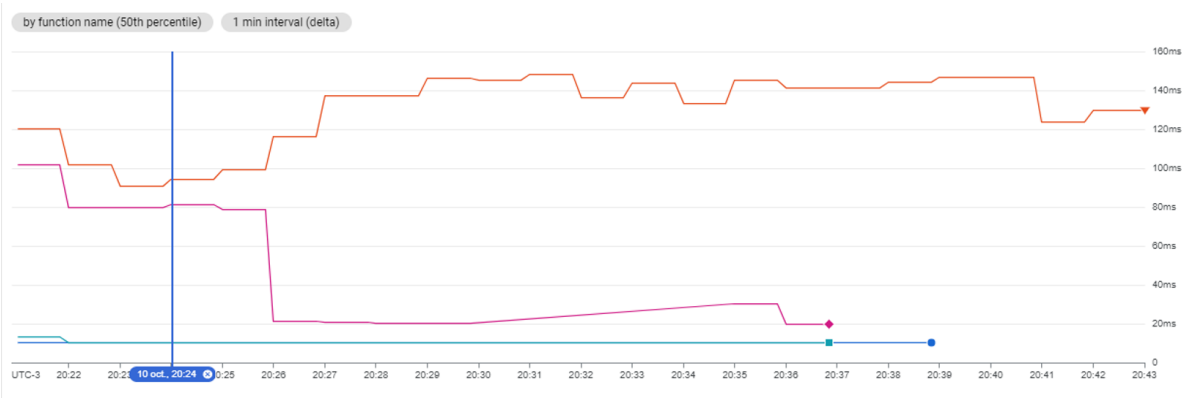
Viendo las estadísticas de tiempos encontramos a un nuevo responsable:

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)
GET	/app?view=about	64	65	67	70	82	110
GET	/app?view=about_legals	64	65	67	69	82	120
GET	/app?view=about_offices	64	65	67	70	80	110
GET	/app?view=home	64	65	67	70	81	120
GET	/app?view=jobs	64	65	67	70	82	120
GET	/get-counter?visit_type=about	61	62	64	67	83	150
GET	/get-counter?visit_type=about_legals	61	62	64	68	92	160
GET	/get-counter?visit_type=about_offices	61	62	64	67	83	150
GET	/get-counter?visit_type=home	61	62	64	67	82	140
GET	/get-counter?visit_type=jobs	61	62	64	67	79	130
POST	/inc-counter?visit_type=about	4300	5500	7700	9500	10000	10000
POST	/inc-counter?visit_type=about_legals	4300	5400	7600	9500	10000	10000
POST	/inc-counter?visit_type=about_offices	4200	5400	7500	9400	10000	10000
POST	/inc-counter?visit_type=home	4200	5500	7600	9400	10000	10000
POST	/inc-counter?visit_type=jobs	4200	5400	7500	9400	10000	10000
	Aggregated	**67**	**76**	**170**	**3000**	**7500**	**9800**

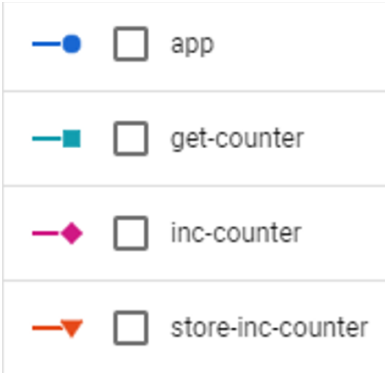
Por un lado, **confirmamos** nuestra hipótesis anterior: *“La baja performance de `get-counter` es debido a que debe obtener y reducir los contadores cada vez que es ejecutada. Podremos optimizar esto con un caché en memoria de los cómputos efectuados”.*
Efectivamente, agregar un cache a get-counter **mejoró sustancialmente** los tiempos de respuesta, moviendo nuestro breakpoint.

Por otro lado, encontramos un nuevo limitante: los tiempos de respuesta de inc-counter, que son entre 2 y 3 órdenes de magnitud mayores a sus contrapartes. Esto genera los failures que vemos en el gráfico, que son nuevamente códigos 429.

Revisando ahora GCP, vemos los siguientes tiempos de ejecución (50% percentil):

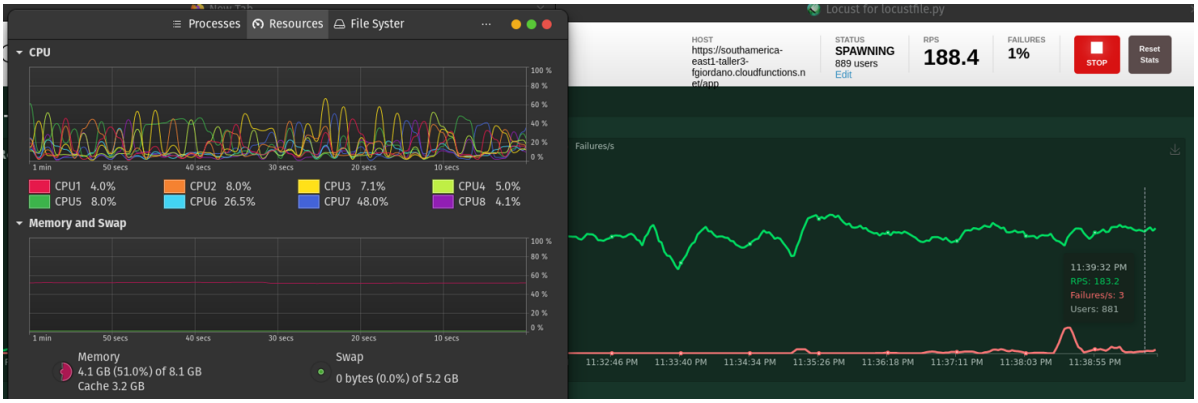


Con las referencias:

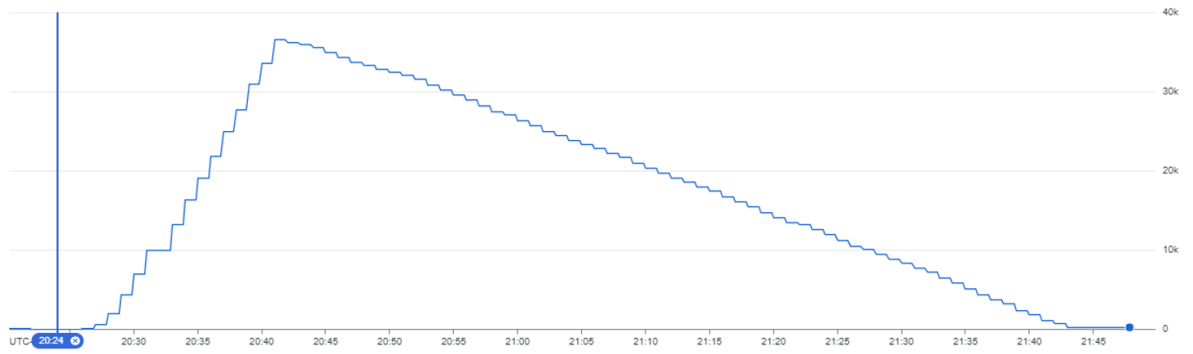


Vemos que inc-counter es efectivamente más lenta que app y get-counter, mientras que store-inc-counter sigue en 150ms, siendo ahora la más lenta de todas.

Tenemos nuevamente discrepancias entre las estadísticas de Locust y GCP, las cuales atribuimos a timeouts internos de GCP. Durante las pruebas, la computadora responsable no agotó sus recursos, por lo que no puede ser un problema local:

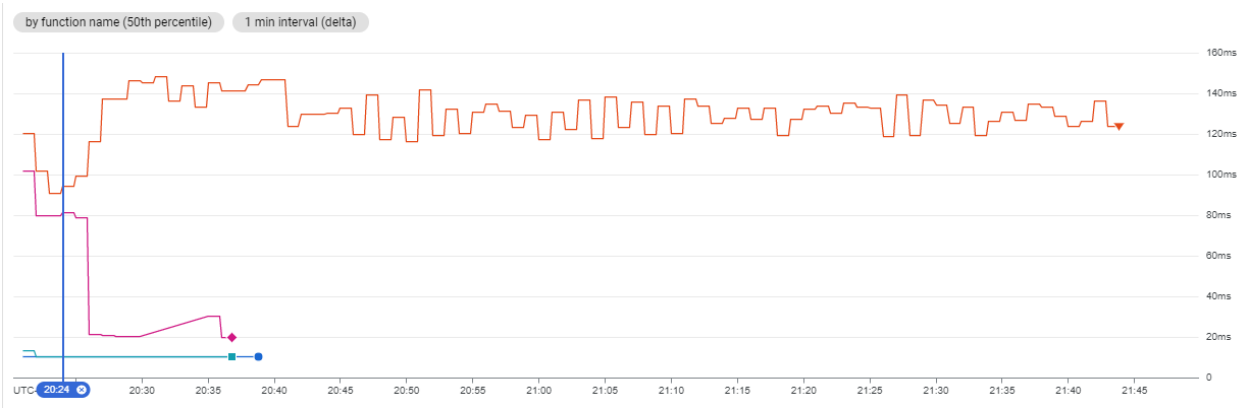


Con todo esto, si bien tenemos problemas con inc-counter, encontramos otros de la mano de store-inc-counter y su cantidad de mensajes sin procesar:

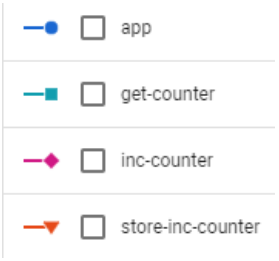


Así, para una carga de 15 minutos, el worker asincrónico tardó **una hora extra** en procesar los incrementos, lo cual ya no parece tan rapido.

Viendo los tiempos de ejecución (50% percentil) para esta hora, notamos que el mismo se mantiene (relativamente) constante:



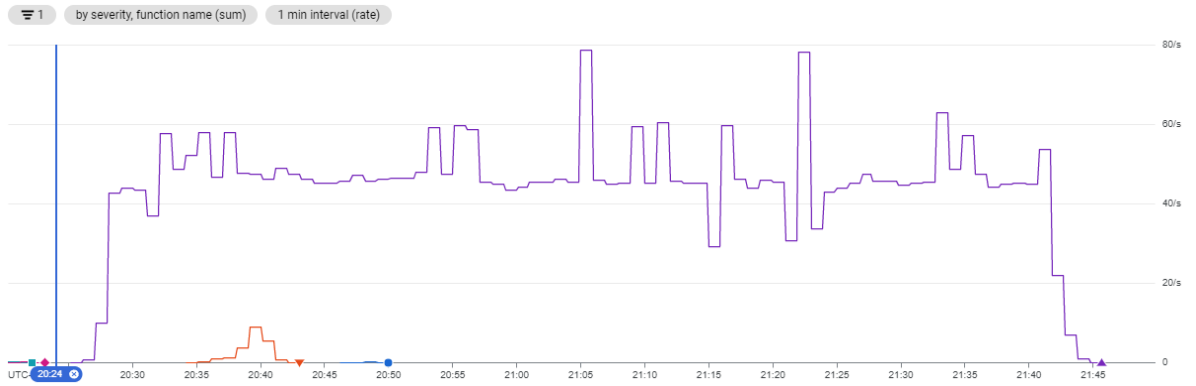
Referencia:



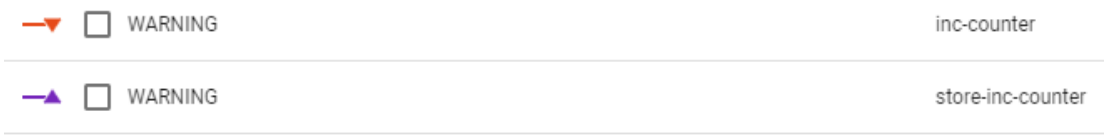
Por lo que esta tardanza no depende de la carga ni de la cantidad de mensajes pendientes. Viendo estadísticas de firestore, vemos que las escrituras oscilan los 10 writes/segundo:

Recordando que tenemos 5 páginas con 2 shards cada una, y que Firestore posee un soft limit de 1 write/segundo por shard, tendríamos un soft limit teórico máximo de 10 writes/segundo. Así, sospechamos que Firestore podría estar limitando la performance de store-inc-counter.

Por último, analizamos la cantidad de WARNINGs código 429 emitidas por cada función por segundo:



Con referencia:



Vemos que store-inc-counter tiene muchas dificultades para procesar a tiempo toda la demanda. Lo mismo para inc-counter, pero en menor medida.

Con estos datos, concluimos dos puntos de mejora:

1. Mejorar los tiempos de inc-counter: probablemente mejore los RPS y la cantidad máxima de usuarios soportados.
2. Mejorar los tiempos de store-inc-counter: permitirá que el sistema se estabilice más rápido, mostrando mejor consistencia en los contadores vistos por los usuarios.

De aquí surgen varios caminos para cada uno:

1. Mejorar los tiempos de inc-counter:

   a. Incrementar la cantidad de instancias: simplemente aumentando la disponibilidad de instancias debería mejorar la disponibilidad percibida por el usuario.
   b. Optimizar el código: de momento, no se observa ninguna optimización obvia del mismo. Podría intentarse batchearse las publicaciones al tópico, pero esto rompería con la propiedad stateless de las funciones serverless, además de que podría fácilmente haber pérdida de mensajes.
2. Mejorar los tiempos de store-inc-counter:
   a. Incrementar la cantidad de shards: Firestore tiene un *soft limit* de 1 write / segundo a cada shard si se quiere tener buena performance. Más shards subirán el límite teórico máximo.
   b. Incrementar la cantidad de instancias: igual que para el punto 1, incrementar la cantidad de workers asincrónicos podría procesar más rápido los mensajes recibidos por el tópico, si es que Firestore no es el limitante.
   c. Optimizar el código: Como en el otro punto, se podrían batchear los pull del tópico y los incrementos a la base de datos. Esto requeriría cambiar el modo de suscripción de la función (de *push* a *pull*) y el manejo de estado dentro de la función, considerando que se pueden perder los mensajes.

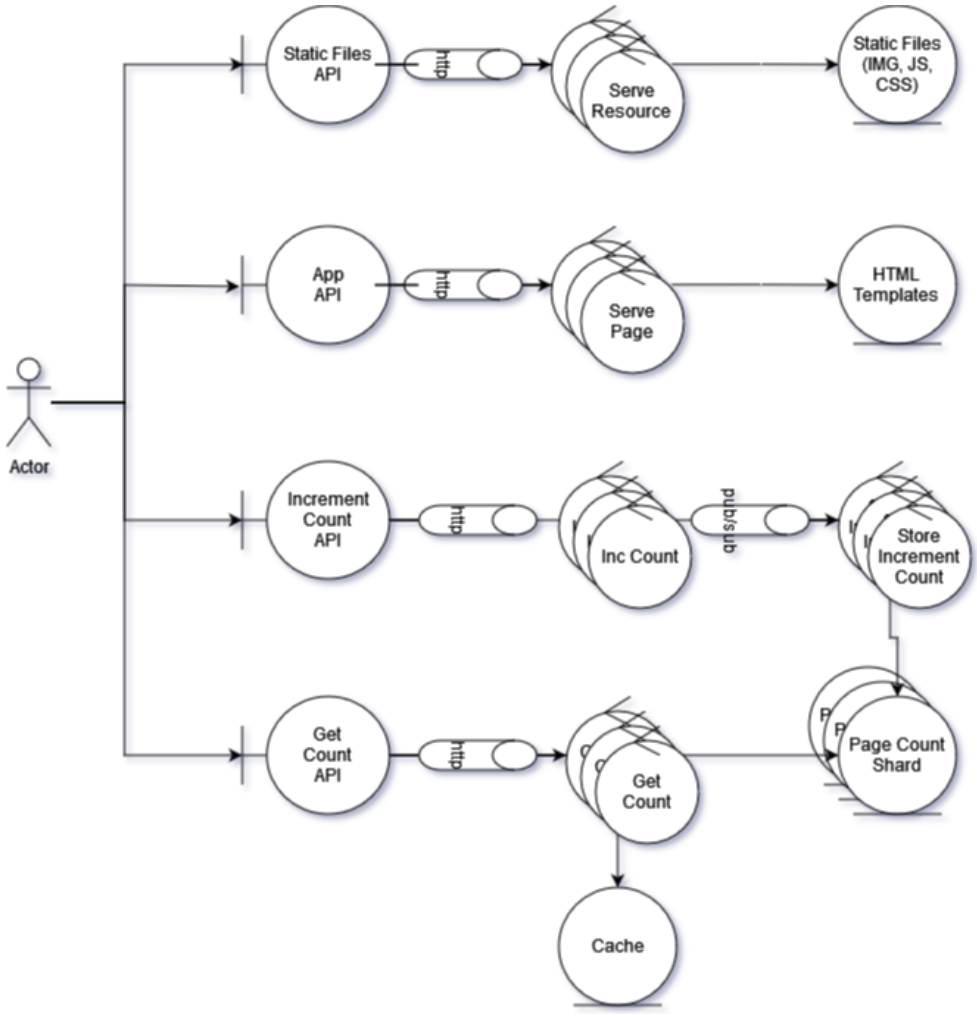
A fin de mejorar la consistencia del sistema antes de que se vuelva un mayor problema, intentaremos seguir por el camino 2, mejorando store-inc-counter. Para ello, comenzaremos probando la propuesta A

Tendremos entonces una nueva hipótesis a comprobar:
“La baja performance de store-inc-counter se debe al soft limit impuesto por Firestore. Para mejorarlo, bastará con aumentar la cantidad de shards por página de 2 a 20, moviendo el nuevo límite a 100 writes por segundo.”

Trabajaremos esto en la siguiente iteración.

Iteración 2

Continuamos con la misma arquitectura:



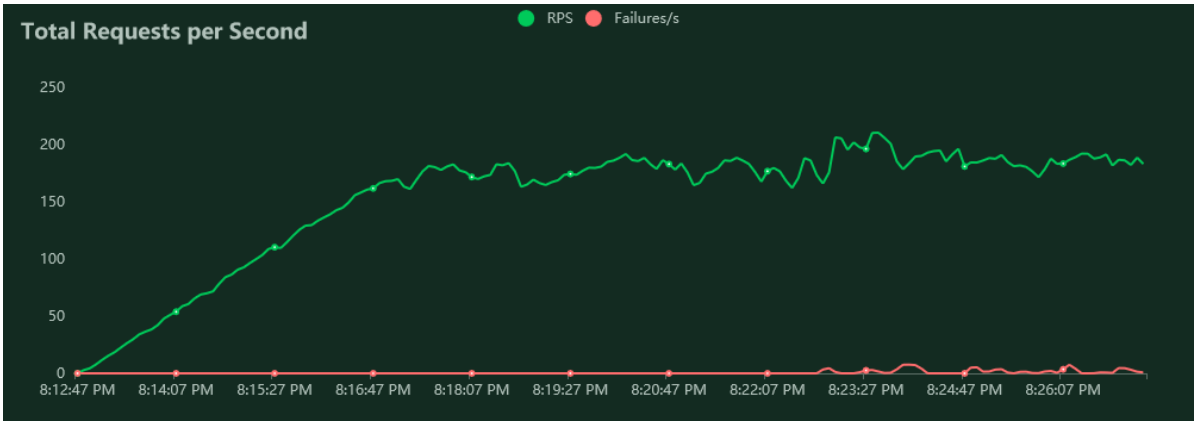
Con la diferencia de que ahora tendremos **20 shards por página**, manteniendo los **10 segundos de timeout de caché**.

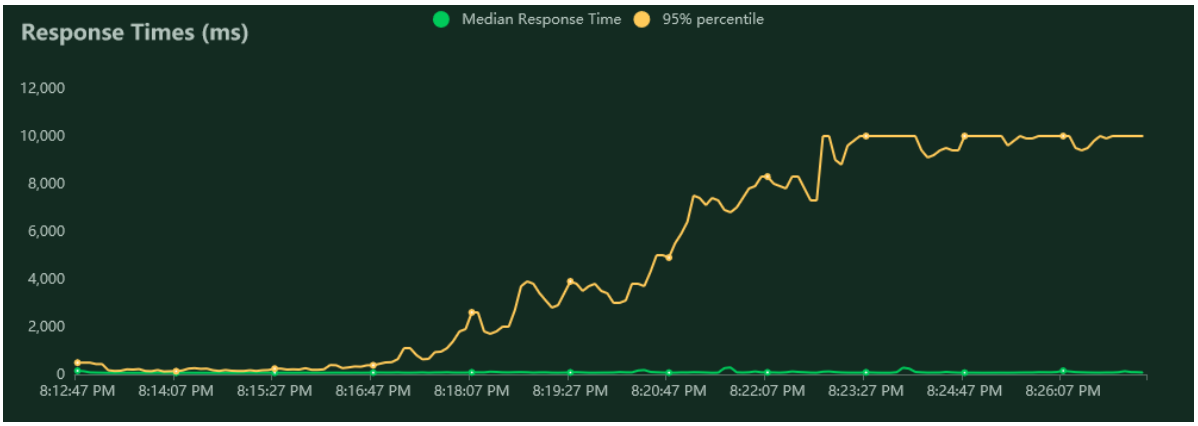
Breakpoint Test

Llevamos el sistema al límite con la misma configuración:

- Cantidad máxima de usuarios: 30000 (“infinito”)
- Incrementos: +1 usuario / segundo
- Tiempo total de corrida: 5:12:45 PM - 5:27:15 PM (15 minutos)
- Límite de instancias: 2 para cada Cloud Function

Desde Locust (vista del usuario), ocurre lo esperado: seguimos con los mismos RPS y fallas:

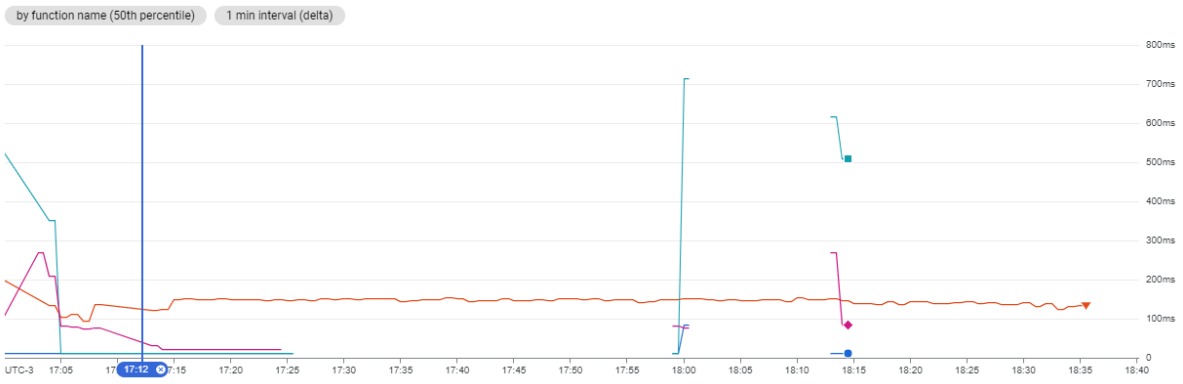




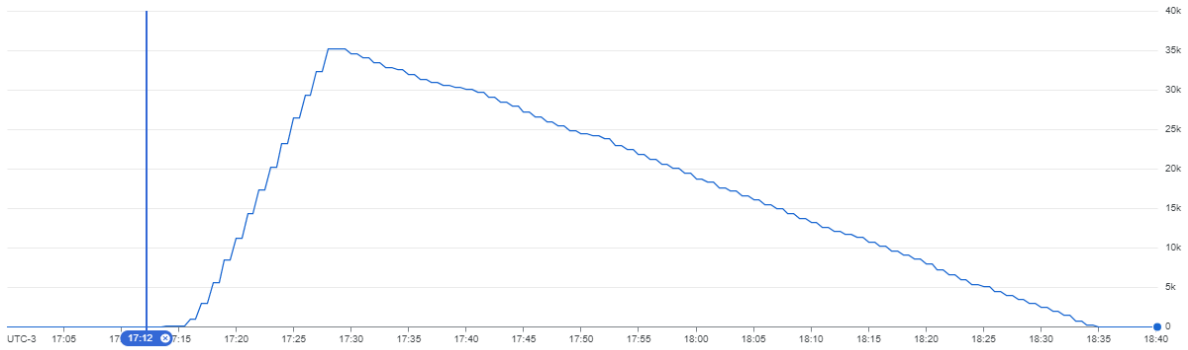
Nuevamente por inc-counter:

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)
GET	/app?view=about	61	63	66	73	97	130
GET	/app?view=about_legals	61	63	66	73	100	130
GET	/app?view=about_offices	61	63	66	73	99	130
GET	/app?view=home	61	63	66	72	98	130
GET	/app?view=jobs	61	63	66	72	96	120
GET	/get-counter?visit_type=about	63	70	87	200	440	690
GET	/get-counter?visit_type=about_legals	63	71	92	260	460	680
GET	/get-counter?visit_type=about_offices	63	68	83	190	420	660
GET	/get-counter?visit_type=home	63	69	87	200	450	660
GET	/get-counter?visit_type=jobs	63	68	83	180	400	620
POST	/inc-counter?visit_type=about	3000	4400	7200	8700	9900	10000
POST	/inc-counter?visit_type=about_legals	2800	4100	7200	8900	9900	10000
POST	/inc-counter?visit_type=about_offices	2800	3800	7100	8600	9800	10000
POST	/inc-counter?visit_type=home	2900	4200	7100	8600	9800	10000
POST	/inc-counter?visit_type=jobs	2900	4100	7200	8600	9800	10000
	Aggregated	**75**	**110**	**410**	**2100**	**7100**	**9400**

De todas formas, esto era esperado, pues lo que queremos mejorar son los tiempos de store-inc-counter. Vemos ahora su tiempo de ejecución (50% percentil) en GCP:



Nos llevamos con esto una sorpresa, pues el tiempo de ejecución de store-inc-counter sigue rondando los 130ms. Estudiamos si es que se procesaron más rápido los mensajes en el tópico:



Vemos que **no** hubo una mejora significativa en la performance. El sistema **sigue tomando una hora extra** para procesar una carga de 15 minutos.

Viendo las estadísticas de Firestore, encontramos que tampoco hubo un cambio en la tasa de escritura:

Sigue rondando los 10 writes/segundo aun teniendo un limite teórico de 100 writes/segundo, por lo que **Firestore no está limitando la performance de store-inc-counter.**

Así, concluimos que nuestra hipótesis: *“La baja performance de store-inc-counter se debe al soft limit impuesto por Firestore. Para mejorarlo, bastará con aumentar la cantidad de shards por página de 2 a 20, moviendo el nuevo límite a 100 writes por segundo”* resulta ser **falsa.**

Observando la tasa de WARNINGs 429 seguimos teniendo una falta de instancias disponibles:

Referencia:

Recordando las mejoras propuestas en la Iteración 1, y teniendo aún baja performance con store-inc-counter, optaremos ahora por incrementar la cantidad de instancias.

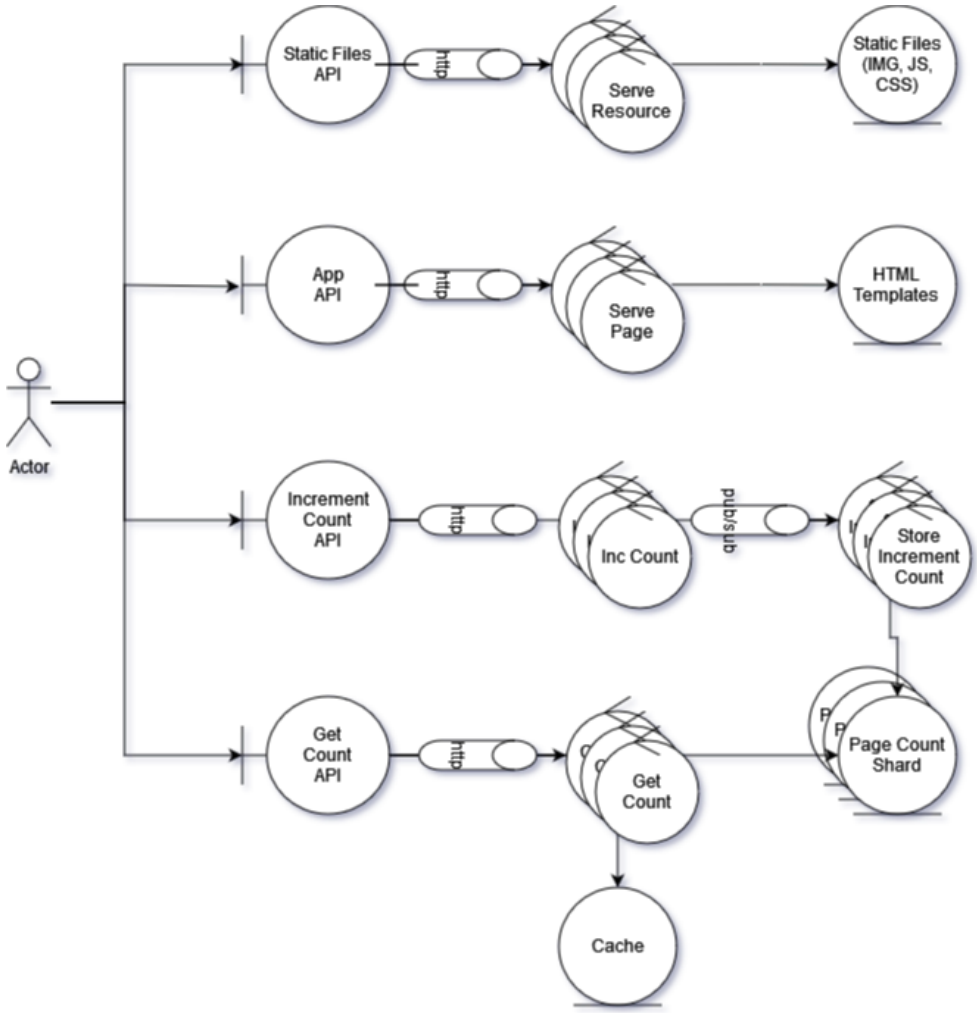
Tendremos ahora la hipótesis:

“Incrementar la cantidad de workers asincrónicos permitirá procesar más rápido los mensajes recibidos por el tópico. Si aumentamos la cantidad de instancias de store-inc-counter de 2 a 4 deberían procesarse todos los mensajes pendientes el doble de rápido.”

Si bien esta Iteración 2 no trajo mejoras, mantendremos los 20 shards por página pues nos asegurará de que Firestore no será un problema para las siguientes etapas. Decidimos esto ya que no hubo cambios significativos en los tiempos de respuesta de funciones como get-counter, que podría haber sido una de las más perjudicadas.

Iteración 3

Seguimos con la misma arquitectura:



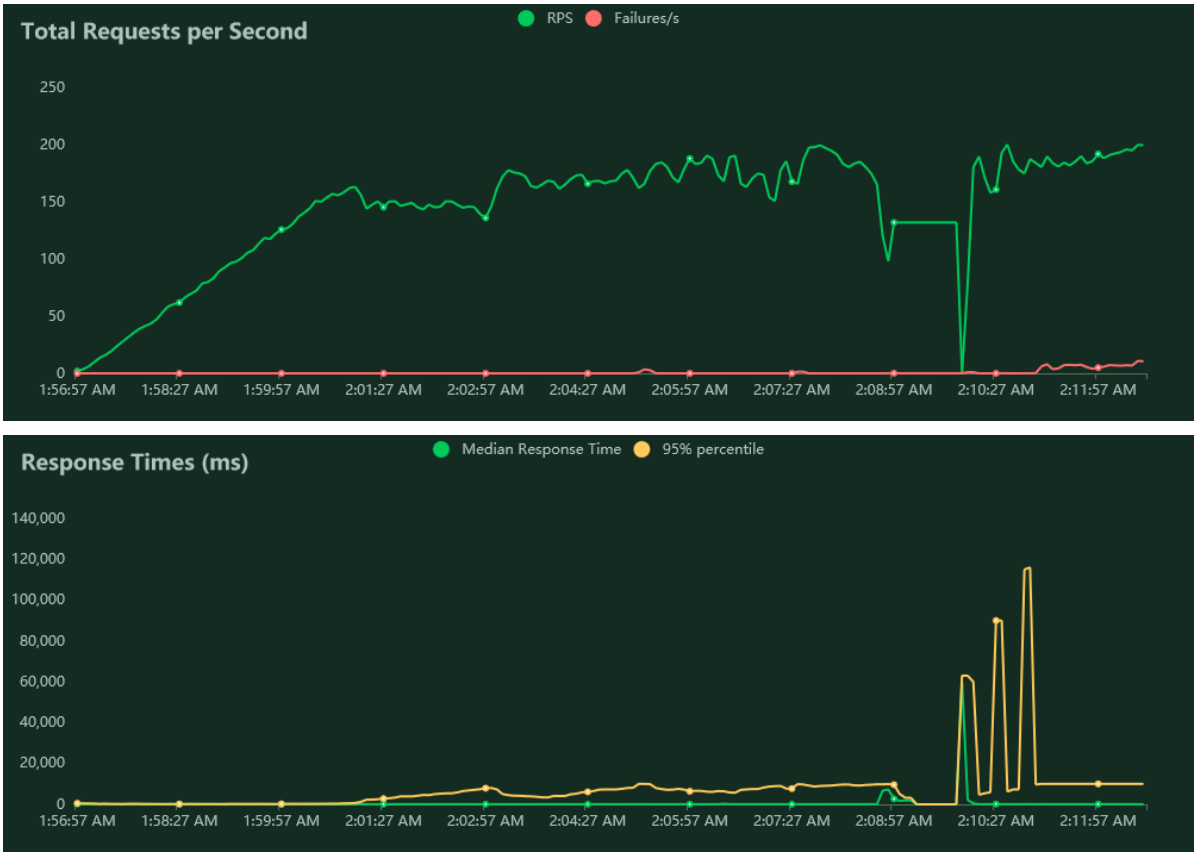
Mantenemos también los **20 shards por página** y **10 segundos de timeout de caché**. La diferencia ahora radica en la cantidad de instancias: **4 instancias para store-inc-counter, y 2 para cada una de las demás**.

Breakpoint Test

Llevamos el sistema al límite con la siguiente configuración:

- Cantidad máxima de usuarios: 30000 (“infinito”)
- Incrementos: +1 usuario / segundo
- Tiempo total de corrida: 10:56:53 PM - 11:12:39 PM (15 min)
- Límite de instancias: 4 para store-inc-counter, 2 para el resto de las Cloud Function.

Siguiendo con la misma carga de usuarios, obtenemos los resultados de Locust. Esperamos que los mismos no cambien pues seguimos con mejoras del worker asincrónico:

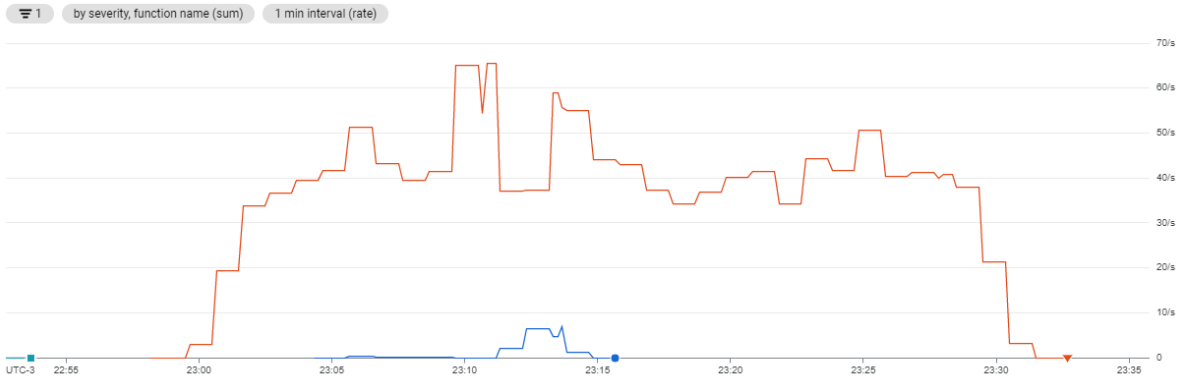


Desafortunadamente, aproximadamente a las 2:08:57, hubo una interrupción en el servicio de internet local de 1 minuto de duración. Este corte produjo también los picos en los tiempos de respuesta a las 2:10:27, donde saltaron algunos timeouts locales (60000 ms). De todas formas, llegando al final, seguimos recibiendo errores 429 por parte de inc-counter.

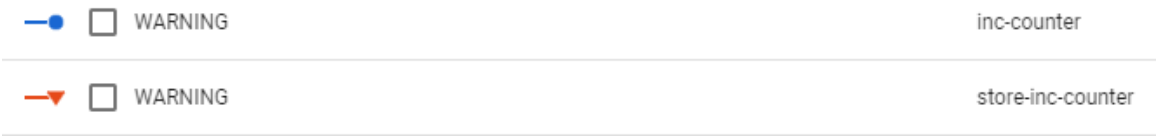
Confirmamos que fue una falla local y no de GCP pues hallamos failures básicos de sockets e internet en Locust:

Method	Name	Error	Occurrences
POST	/inc-counter?visit_type=jobs	429 Client Error: Too Many Requests for url: https://southamerica-east1-taller3-fgiordano.cloudfunctions.net/inc-counter?visit_type=jobs	279
POST	/inc-counter?visit_type=about_legals	429 Client Error: Too Many Requests for url: https://southamerica-east1-taller3-fgiordano.cloudfunctions.net/inc-counter?visit_type=about_legals	66
GET	/get-counter?visit_type=jobs	[Errno 104] Connection reset by peer	1
POST	/inc-counter?visit_type=home	429 Client Error: Too Many Requests for url: https://southamerica-east1-taller3-fgiordano.cloudfunctions.net/inc-counter?visit_type=home	181
GET	/app?view=jobs	[Errno 104] Connection reset by peer	1
GET	/app?view=about_offices	[Errno -3] Temporary failure in name resolution	2
GET	/app?view=home	[Errno -3] Temporary failure in name resolution	3
POST	/inc-counter?visit_type=about_offices	429 Client Error: Too Many Requests for url: https://southamerica-east1-taller3-fgiordano.cloudfunctions.net/inc-counter?visit_type=about_offices	113
POST	/inc-counter?visit_type=about	429 Client Error: Too Many Requests for url: https://southamerica-east1-taller3-fgiordano.cloudfunctions.net/inc-counter?visit_type=about	104
GET	/app?view=jobs	[Errno -3] Temporary failure in name resolution	2
GET	/app?view=about	[Errno -3] Temporary failure in name resolution	3

Pero notamos la ausencia de los mismos en GCP, teniendo solo WARNINGs 429:



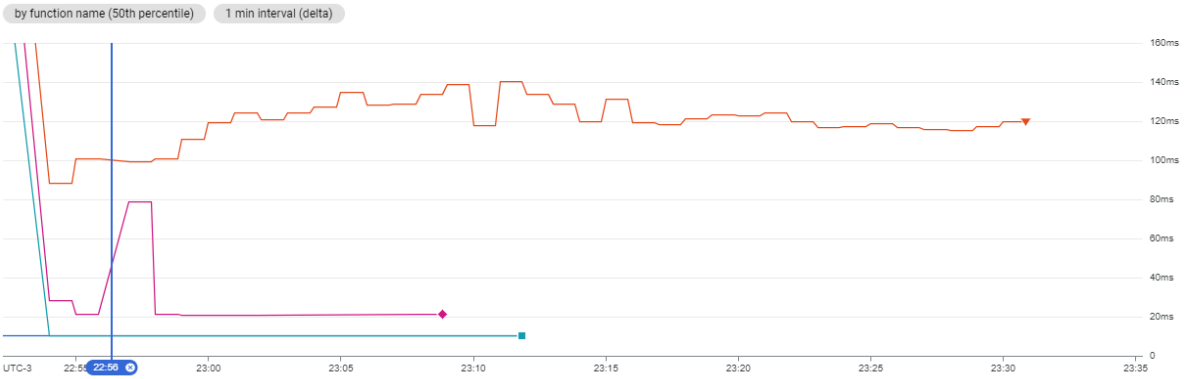
Referencia:



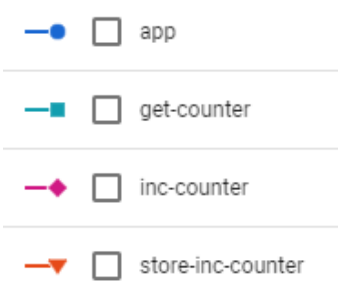
Más allá de esta breve interrupción, no afectó a las métricas de tiempos:

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)
GET	/app?view=about	65	66	69	73	96	140
GET	/app?view=about_legals	65	66	69	73	98	150
GET	/app?view=about_offices	65	67	69	73	97	140
GET	/app?view=home	65	67	69	73	100	140
GET	/app?view=jobs	65	67	69	74	100	140
GET	/get-counter?visit_type=about	64	66	72	120	380	760
GET	/get-counter?visit_type=about_legals	64	67	75	170	400	710
GET	/get-counter?visit_type=about_offices	63	66	71	100	380	700
GET	/get-counter?visit_type=home	63	66	72	100	370	690
GET	/get-counter?visit_type=jobs	63	66	71	98	360	680
POST	/inc-counter?visit_type=about	4300	5800	7100	9100	9900	10000
POST	/inc-counter?visit_type=about_legals	4300	5900	7100	9100	10000	10000
POST	/inc-counter?visit_type=about_offices	4200	5800	7100	9100	9900	10000
POST	/inc-counter?visit_type=home	4200	5600	7000	9000	9900	10000
POST	/inc-counter?visit_type=jobs	4300	5800	7100	9100	10000	10000
	Aggregated	**72**	**90**	**300**	**3300**	**7200**	**9800**

Ni mucho menos los tiempos de ejecución (50% percentil) de GCP:

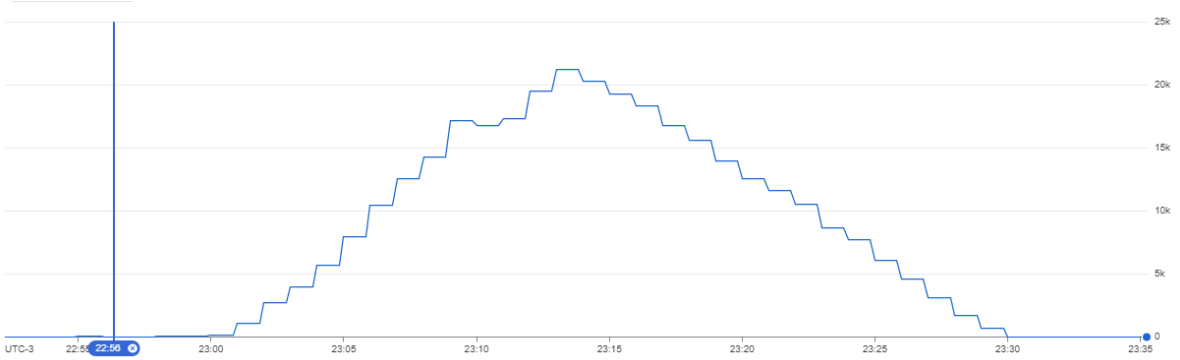


Referencia:



Vemos que aumentar la cantidad de instancias **no** afecta el tiempo de ejecución de store-inc-counter, siguiendo en ~130ms.

Recordando nuestra hipótesis, queremos analizar el tiempo hasta completar el procesamiento de todos los mensajes. Viendo la cantidad de mensajes pendientes en el tópico, tenemos una pequeña meseta a las 23:10, correspondiente con el corte de internet local:



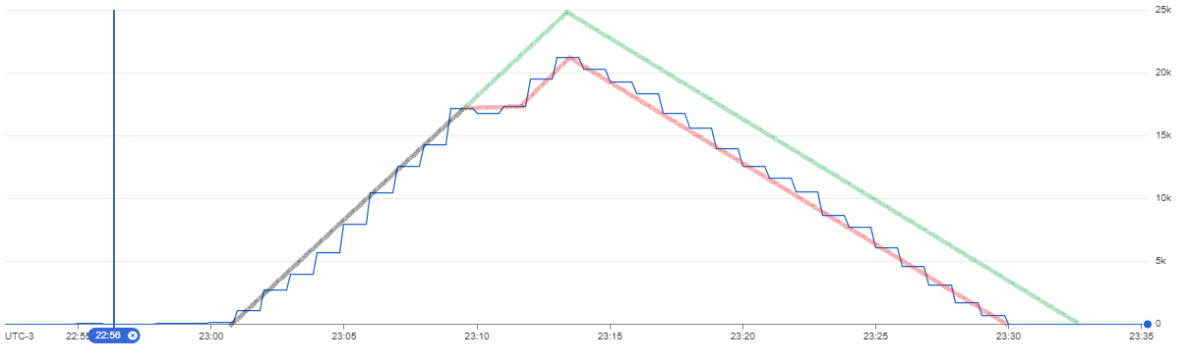
Esto significa que algunos mensajes jamás llegaron a GCP durante 60 segundos.

Teniendo esto en cuenta, intentaremos ahora compensarlo para poner a prueba nuestra hipótesis. Aproximando el throughput de store-inc-counter como lineal, estudiamos ahora la pendiente con la que decrecen la cantidad de mensajes pendientes.

Siendo que tenemos un pico de mensajes aproximadamente a las 23:13PM con ~21.21k mensajes y desciende a 0 mensajes para las 23:30PM, tendremos una pendiente de:

$$\frac{0 - 21210}{(30 - 13)*60} = -20.79\ mensajes\ /\ segundo$$

Si esta aproximación no nos satisface, podemos ajustar la cantidad de mensajes pendientes extrapolando a partir de la pendiente de crecimiento:



Teniendo ahora que alcanzamos un pico de 25k mensajes a las 23:13 PM, y completamos a las 23:33PM (1.33 minutos extras por cada minuto de carga):

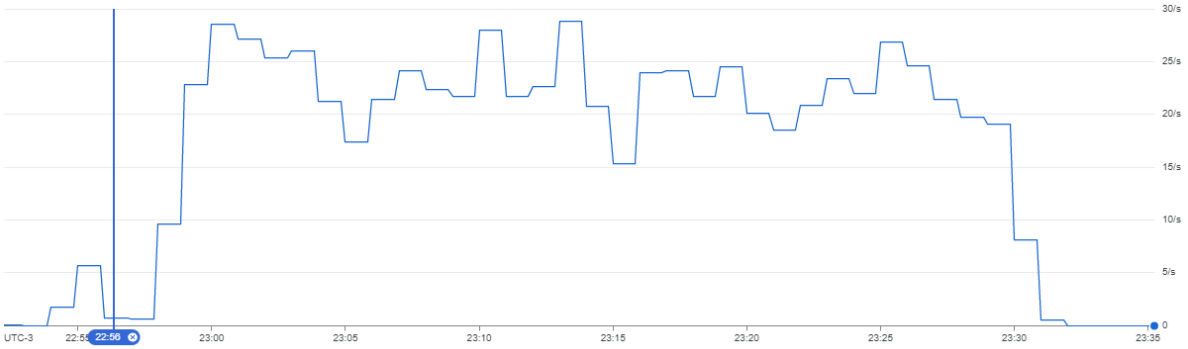
$$\frac{0 - 25000}{(33 - 13)*60} = -20.83\ mensajes\ /\ segundo$$

Calculamos ahora la pendiente de la iteración previa, donde pasamos de 35k mensajes pendientes a las 17:27 hasta completar todos a las 18:35:

$$\frac{0 - 35000}{68*60} = -8.57\ mensajes\ /\ segundo$$

Con estos datos concluimos que la hipótesis: "*Si aumentamos la cantidad de instancias de store-inc-counter de 2 a 4 deberían procesarse todos los mensajes pendientes el doble de rápido"* es efectivamente **verdadera.** Así, podemos aproximarnos a que para una carga de 15 minutos, el sistema alcanza la consistencia en tan solo 20 minutos extras, lo cual resulta aceptable.

Curiosamente, revisando las escrituras por segundo a Firestore, vemos que fue positivo mantener los 20 shards por página:



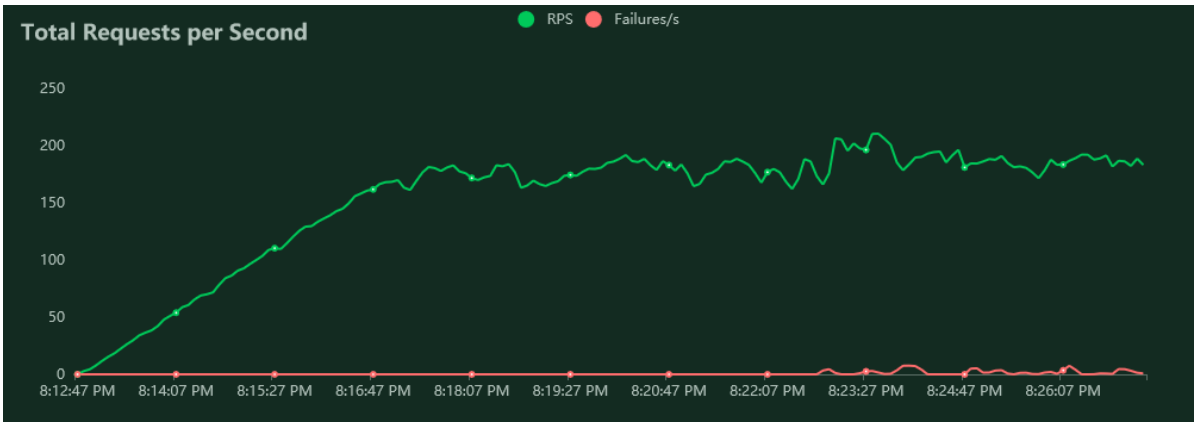
Si bien seguimos muy lejos del límite teórico (100wr/seg), estamos ahora aprovechando y escribiendo a más de 10 writes por segundo.

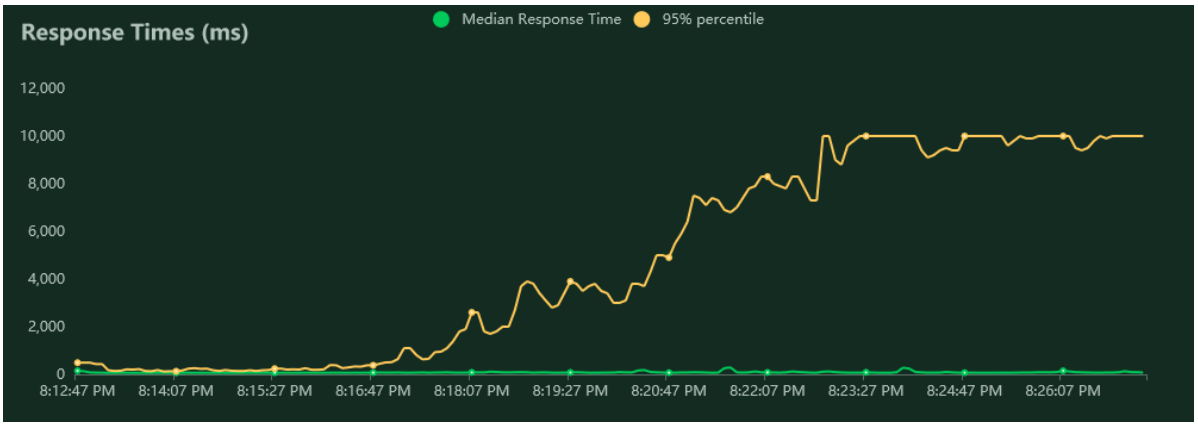
De querer mejorar aún más el sistema, el siguiente paso sería atacar los tiempos de inc-counter como muestra Locust.

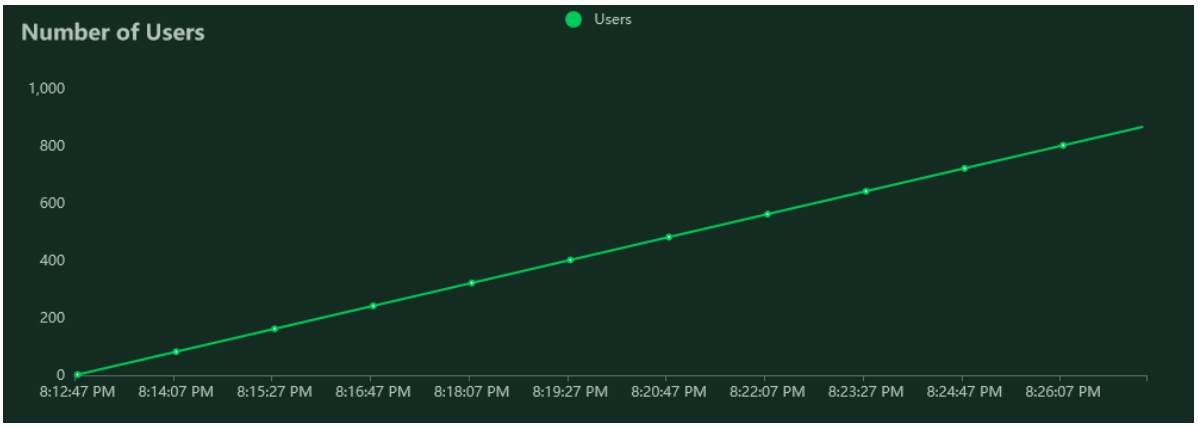
De momento nos detendremos aquí, tomando esta Iteración 3 como la arquitectura final. Aprovecharemos ahora para analizar el comportamiento del sistema con más profundidad.

Endurance Test

La propuesta será ahora mantener una carga constante de a partes sobre el sistema, cerca del breakpoint hallado. Recordando los gráficos de *iteraciones anteriores*:





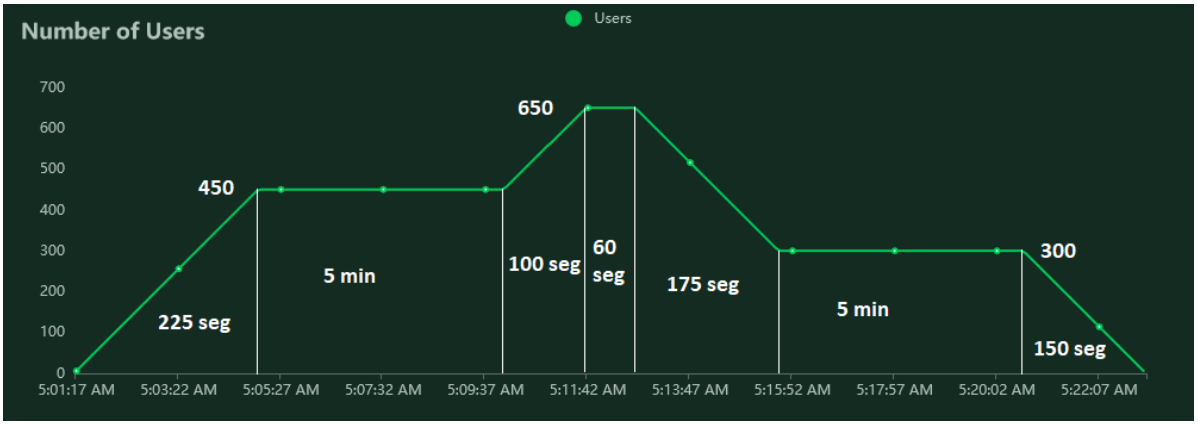


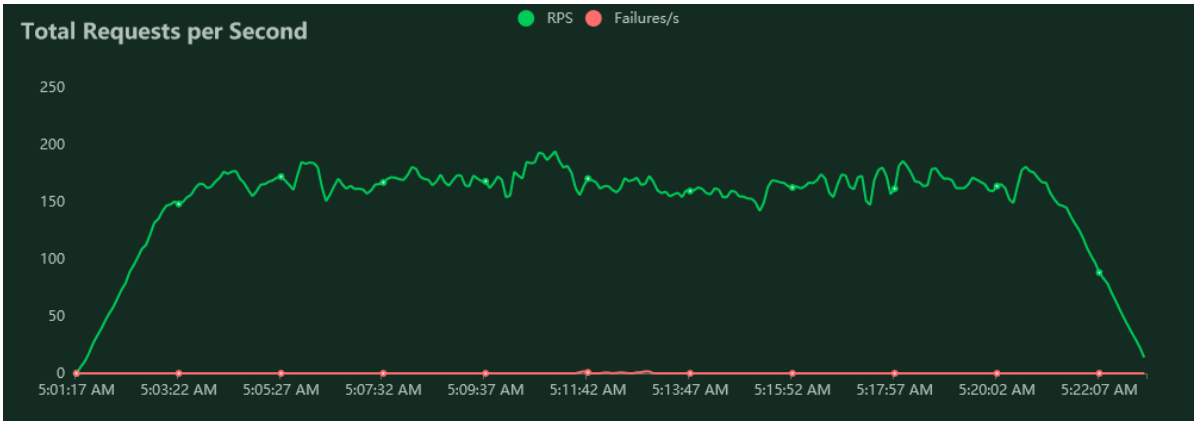
Vemos que se detiene el crecimiento lineal de RPS con ~300 usuarios, y comenzamos a tener grandes timeouts y errores con ~600 usuarios. Por ello, planteamos una carga escalonada que transite estos estados, que presentamos más adelante con los resultados.

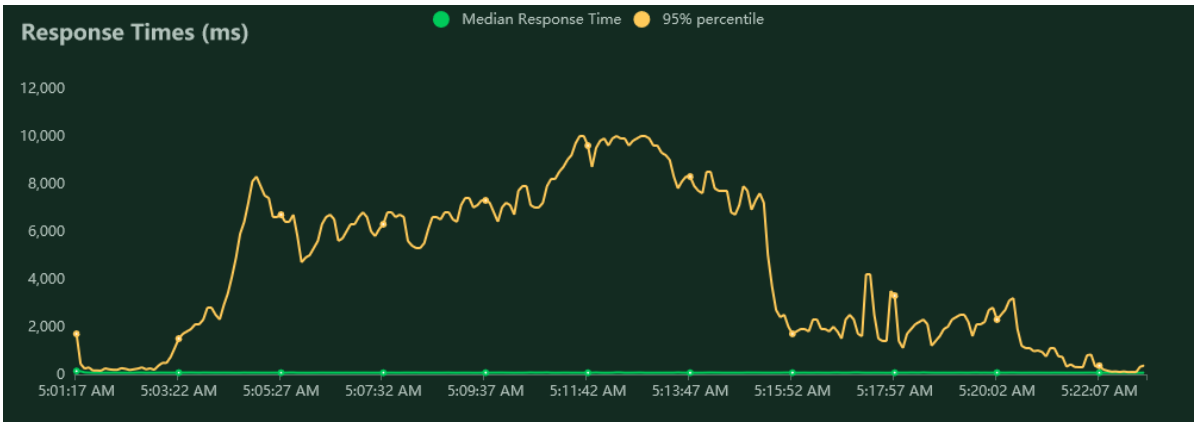
Antes de comenzar la prueba de carga el sistema ya estaba “en caliente” para evitar cold-starts. El resto de las configuraciones son:

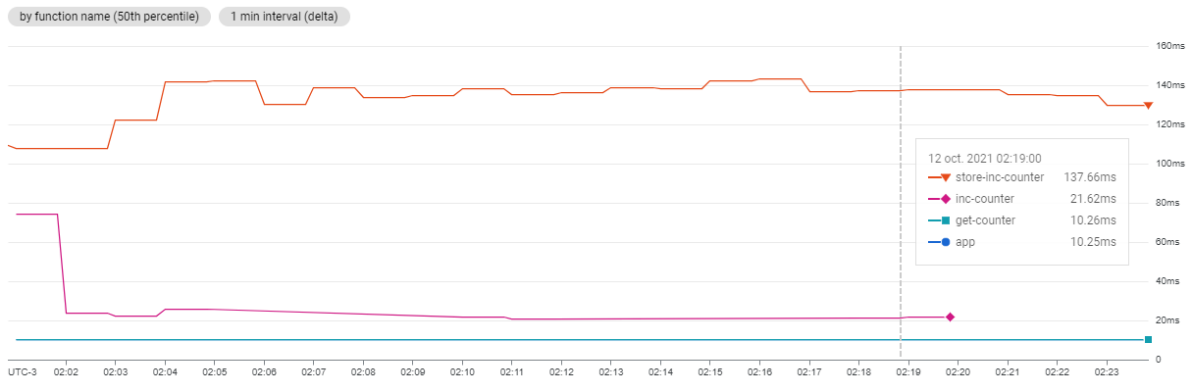
- Cantidad máxima de usuarios: variable, ver más abajo.
- Incrementos/Decrementos: +2 o -2 por segundo, para evitar problemas de escalamiento.
- Tiempo total de corrida: 2:01:14 AM - 2:23:03 AM (22 min)
- Límite de instancias: 4 para store-inc-counter, 2 para el resto de las Cloud Function.

La prueba realizada es la que se presenta a continuación, junto con las métricas alineadas en escala temporal:

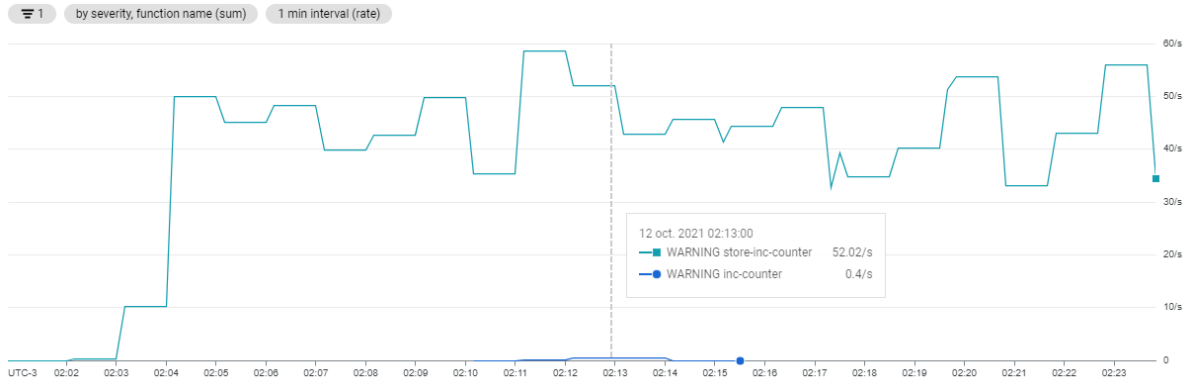






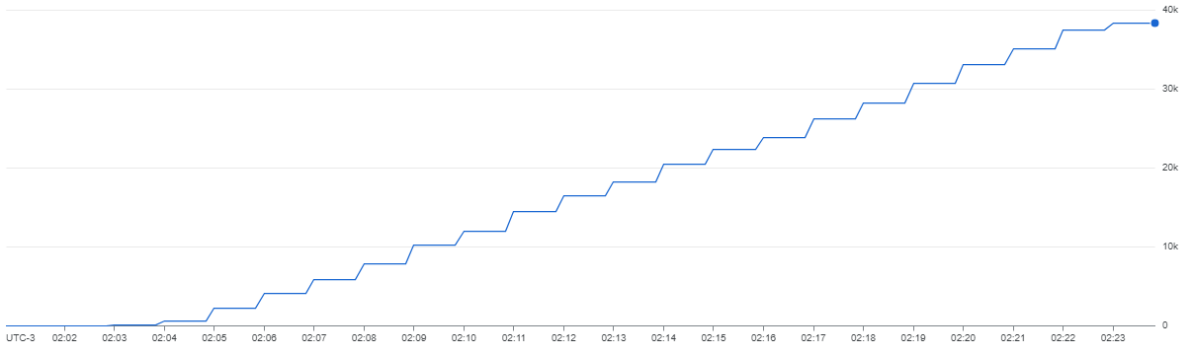


Tiempos de ejecución (50% percentil)



Warnings 429

Escrituras por segundo en Firestore



Mensajes pendientes en el tópico

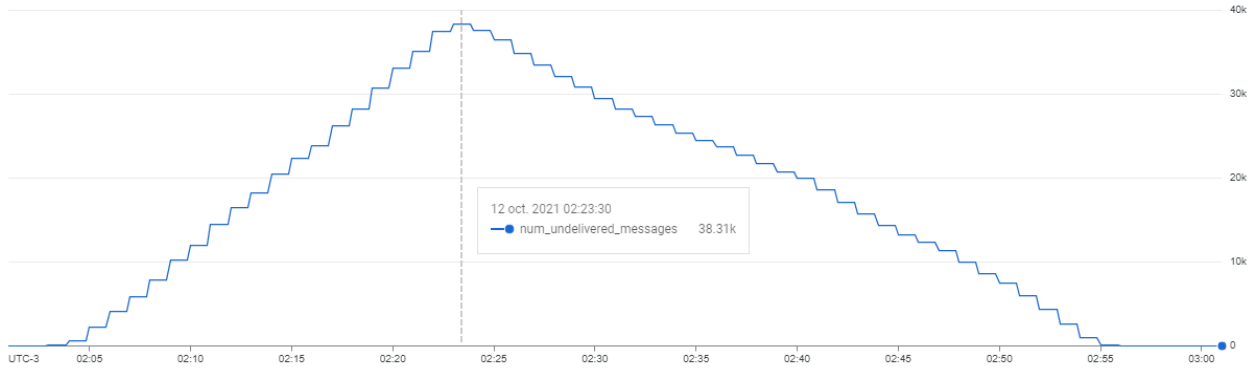
Notamos varios puntos de interés:

- Nuevamente, los RPS siguen constantes en ~175 siempre y cuando se esté por encima de los 280-300 usuarios, sin importar si los usuarios crecen, decrecen o se mantienen.
- inc-counter comienza a fallar con +600 usuarios, pero vuelve rápidamente a su estado anterior (respuestas lentas pero sin fallas) si decrece la carga.
- Con +315 usuarios, inc-counter inmediatamente arroja respuestas mucho más lentas, dando un 95% percentil de 6 segundos o más. Al decrecer, recupera inmediatamente un tiempo rápido de respuesta.
- Los tiempos de ejecución siguen siempre constantes, sin importar si sube o baja la carga.
- Lo mismo ocurre con los WARNINGs 429 de store-inc-counter que siguen constantes. Mientras, inc-counter los arroja solo durante el pico de 650 usuarios, pero rápidamente se recupera.
- Las escrituras por segundo a Firestore oscilan siempre los 23 writes/segundo, independiente de la carga.
- La cantidad de mensajes pendientes parece crecer siempre lineal, sin importar la carga (aun con 300 usuarios o menos). Esto parece indicar que inc-counter tiene un throughput máximo que no se adapta a la carga.

Para esto último, podemos encontrar el throughput máximo observando los publish por segundo en el tópico:

Vemos que oscila unos 55 push / segundo de forma constante. Siendo que inc-counter posee 2 instancias, podemos estimar su throughput como ~27 ejecuciones / segundo.

Nos queda por último observar store-inc-counter, en particular cuanto tardo en procesar todo. Graficamos los mensajes pendientes en el tópico, ahora con una escala temporal extendida:



Con una carga de 22 minutos, los workers lograron procesar su totalidad con 32 minutos extras. (1.45 min extra por cada min de carga)

Conclusiones

Habiendo completado estos últimos análisis, ya terminamos de estudiar las principales limitaciones del sistema, tanto externas al cliente como internas en tópicos.

Nos llevamos entonces una arquitectura capaz de:

- Asegurar ~175 RPS para 315 usuarios o menos, sin fallas (al cliente) y rápidas respuestas.
- Asegurar ~175 RPS para 500 usuarios o menos, pero con tiempos de respuesta degradados sólo para incrementar el contador.
- Asegurar ~175 RPS para 650 usuarios o menos, pero presentando los primeros errores solo para incrementar el contador.
- Recuperarse rápidamente de los errores y las degradaciones si es que baja la carga.
- No asegura una pronta consistencia de los contadores durante altas cargas (+20 RPS). Se estima que por cada minuto de carga, el sistema tarda ~1.3 minutos extras en converger al valor final. (ej: con 15 minutos de carga tardará ~20 min extras en converger).

Para estudiar el diseño más a fondo, ver el documento de arquitectura que acompaña este reporte.

A futuro, sería valioso explorar los siguientes caminos con más pruebas:

- Encontrar los RPS soportados por store-inc-counter: la cantidad de mensajes pendientes siempre aumentó descontroladamente. Se sospecha que la misma ronda los 5 RPS, pero debería confirmarse con pruebas más pequeñas.
- Mejorar los tiempos de inc-counter: como primer intento, pueden aumentarse la cantidad de instancias, pues es la principal limitante según Locust.
- Estudiar el efecto de los tiempos de cache: Con 10 segundos se obtuvieron muy buenos resultados pero no se evaluó su influencia con distintos valores.

Anexo - Configuración de Locust

Todos los tests realizados utilizaban la misma simulación de usuario, el cual visitaba todas las páginas web buscando el HTML y los contadores de visitas. Por ejemplo, para `home`:

```python
@task
def get_home(self):
    # grab html
    self.client.get(".../app?view=home")

    # grab stats
    self.client.post('.../inc-counter?visit_type=home')
    self.client.get('.../get-counter?visit_type=home')
```

Los pesos de cada página fueron los siguientes:

- 25.0% get_home
- 33.3% get_jobs
- 16.7% get_about
- 16.7% get_about_offices
- 8.3% get_about_legals

Entre cada task se optó por un tiempo de espera aleatorio de entre 3 y 5 segundos. Las pruebas fueron corridas localmente *sin* una configuración master-slave, pues jamás se vio un uso intensivo de recursos.