PROFILING

JUAN FERNÁNDEZ FERNÁNDEZ
ESTEFANÍA GANFORNINA TRIGUERO
JAVIER GARCÍA CERRADA
FRANCISCO PEREJÓN BARRIOS
FERNANDO ROMERO RIOJA

Contents

Profiling Poemist API

Como vimos en varias de las pruebas de rendimiento realizadas por los miembros del equipo de desarrollo en diversos equipos con distintos componentes cada uno, se observa que en general el rendimiento de la aplicación es bastante bueno. Sin embargo, en los escenarios en los que hay que pasar por el buscador de libros, el rendimiento baja drásticamente y se obtienen unos resultados paupérrimos en comparación con el resto de escenarios. Nuestra sospecha es que este problema tiene que ver con la API que hemos utilizado para los poemas, a continuación, veremos si esta sospecha tiene o no fundamento.

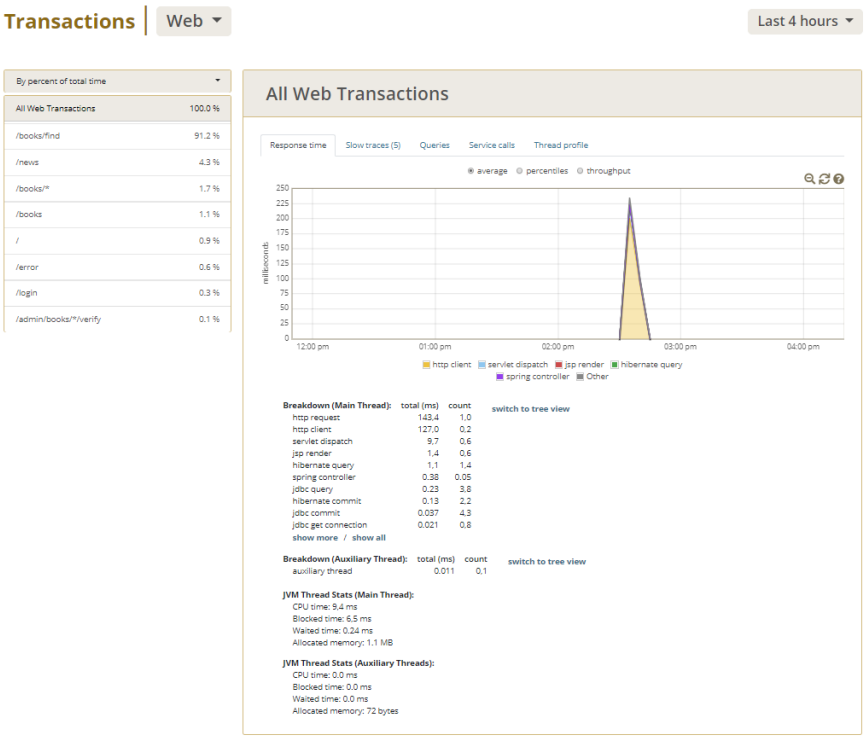
En concreto, vamos a realizar el profiling utilizando el código del Load Test de la Historia de Usuario 04: Verificar libro. En esta prueba, tenemos 2 escenarios:

Escenario positivo: va a home, se loguea, va al buscador de libros donde hace la llamada a la API, obtenemos lista de libros buscamos un libro no verificado y lo verificamos.

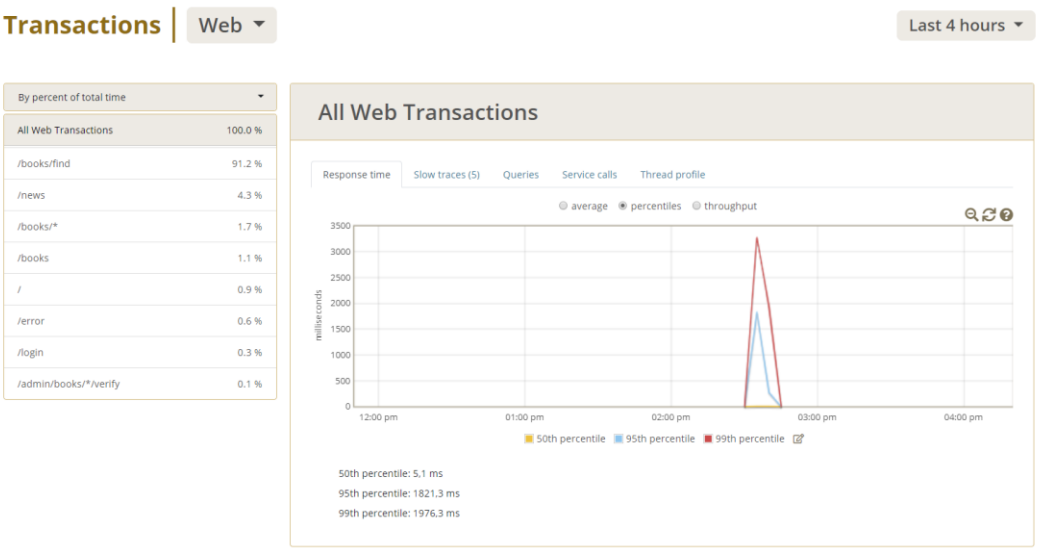
Escenario negativo: sigue los mismos pasos, pero cuando obtiene la lista de libros se mete en uno que ya esté verificado y no hace nada más porque el botón para verificar en ese caso sería falso.

La prueba se realizaría con 35 usuarios durante 100 segundos en ambos escenarios, lo cual creemos que se puede acercar a la cifra real una vez el sistema esté en producción.

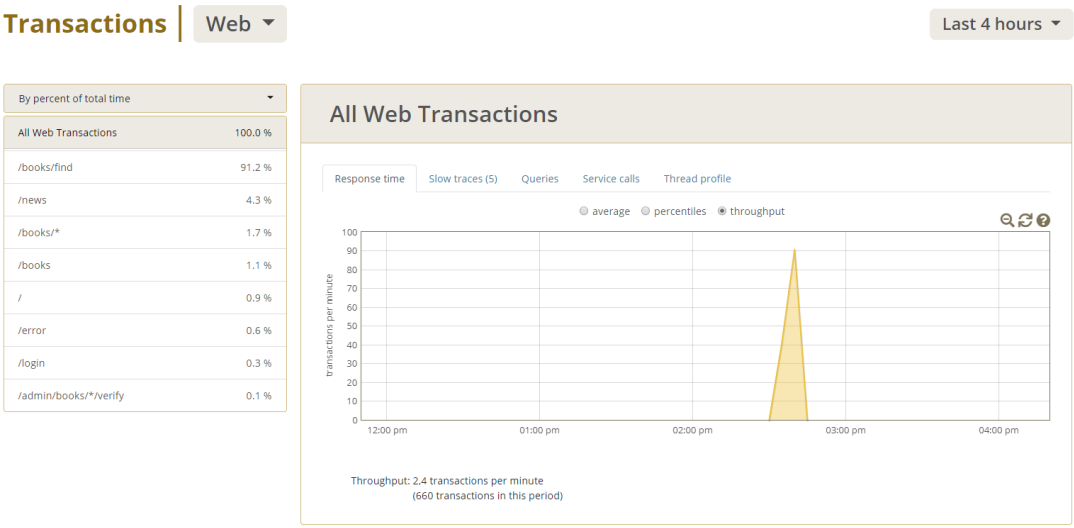
A continuación, se adjuntan capturas del análisis que ha hecho Glowroot habiendo ejecutado nosotros el código del Load Test especificado:



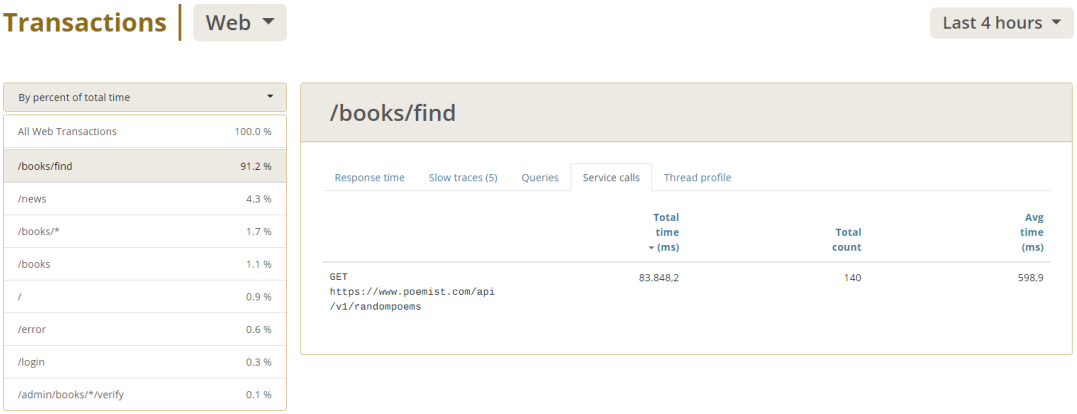
All Web Transactions-Average



All Web Transactions-Percentiles



All Web Transactions-Throughput



Peticiones de servicio

Entre los datos proporcionados se encuentran la razón de peticiones por minuto, el porcentaje de peticiones que se han hecho o el tipo de transacciones que se hacen en cada petición junto con los tiempos empleados en ellas y la media de cuantas veces se realiza cada una.

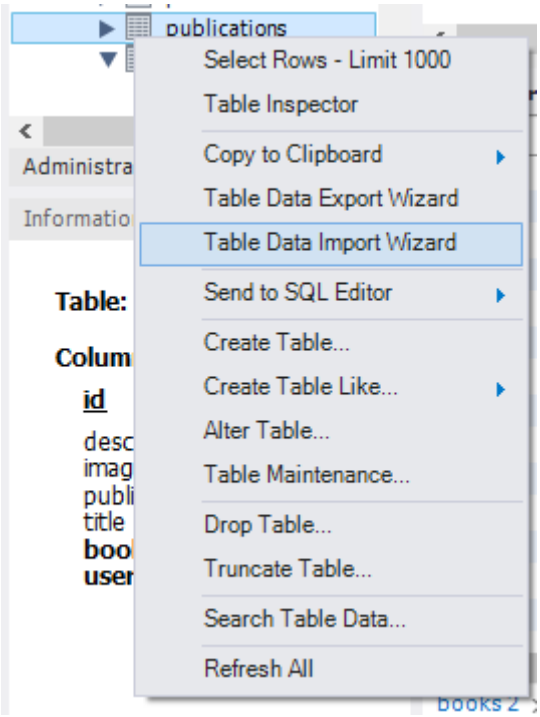
Sin embargo, el dato verdaderamente alarmante aquí es en las peticiones de servicio que, en este caso, se realizan a la API de poemas y que como vemos se emplea un tiempo total de 84 segundos para resolver 140 peticiones, es decir a una media de 0,6 segundos aproximadamente, lo cual es una cifra demasiado elevada en comparación con el resto de tiempos obtenidos.

Por tanto, llegamos a la conclusión de que efectivamente el cuello de botella se produce en las llamadas a Poemist API, confirmándose nuestras sospechas.

La solución a este problema será realizar una refactorización que vendrá detallada en el informe correspondiente.

Profiling HU-07

Hemos poblado las publicaciones con un csv mediante una opcion de mysql

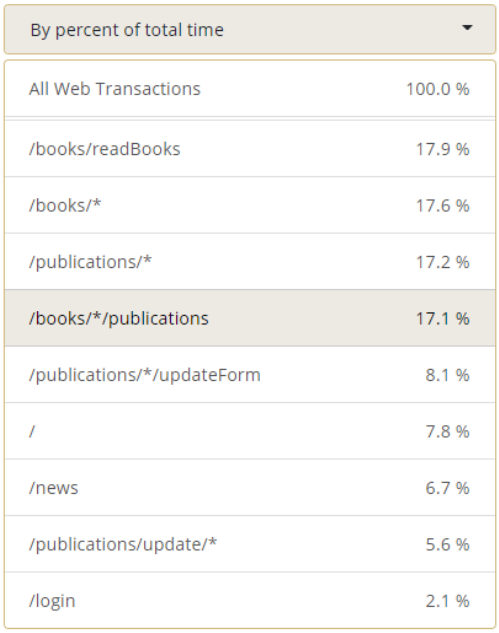


Y aquí vemos un ejemplo de los datos introducidos

79	Esto es una descripcion de prueba el libro esta ...	https://example.jpg	NULL	Correy	1	admin1
80	Esto es otra descripcion de prueba el libro esta ...	https://example.jpg	NULL	Dulcinea	1	admin1
81	Esto es una descripcion de prueba el libro esta ...	https://example.jpg	NULL	Asia	1	admin1
82	Esto es una descripcion de prueba el libro esta ...	https://example.jpg	NULL	Arlina	1	admin1
83	Esto es otra descripcion de prueba el libro esta ...	https://example.jpg	NULL	Lolita	1	admin1
84	Esto es una descripcion de prueba el libro esta ...	https://example.jpg	NULL	Annecorinne	1	admin1
85	Esto es una descripcion de prueba el libro esta ...	https://example.jpg	NULL	Adriana	1	admin1
86	Esto es otra descripcion de prueba el libro esta ...	https://example.jpg	NULL	Glynnis	1	admin1
87	Esto es otra descripcion de prueba el libro esta ...	https://example.jpg	NULL	Rubie	1	admin1
88	Esto es una descripcion de prueba el libro esta ...	https://example.jpg	NULL	Laurene	1	admin1

Ahora procedemos a ejecutar el script con 30 usuarios.

Podemos ver las transacciones por porcentaje del tiempo total de mayor a menor



By percent of total time	
All Web Transactions	100.0 %
/books/readBooks	17.9 %
/books/*	17.6 %
/publications/*	17.2 %
/books/*/publications	17.1 %
/publications/*/updateForm	8.1 %
/	7.8 %
/news	6.7 %
/publications/update/*	5.6 %
/login	2.1 %

Y vemos que está todo más o menos uniforme. Vemos que las 4 primeras son las que requieren más tiempo.

Para /books/readBooks:

Vemos las queries

	Total time (ms)	Total count	Avg time (ms)	Avg rows
`select book0_.id as id1_2_, book0_.isbn as isbn2_2_, book0_.author as author3_2_, book...`	87,9	180	0.49	1,0
`select user0_.username as username1_16_0_, user0_.enabled as enabled2_16_0_, user0_.pa...`	27,6	60	0.46	1,0
`select genre0_.id as id1_3_, genre0_.name as name2_3_ from genres genre0_ order by gen...`	25,8	30	0.86	24,0
`select readbook0_.book_id as col_0_0_ from read_book readbook0_ where readbook0_.user_...`	21,9	30	0.73	6,0

Esta página se trae todos los datos de todos los libros que tengas en leídos, en esta página dado que son libros que ya conoces podríamos dar una lista más simplificada, hacer lo visto en el video de “projections” y mostrar el nombre, el autor y la sinopsis, de esta forma se ahorraría 9 atributos, además de las dos siguientes queries

En books/*

Podemos ver un gran número de queries

	Total time (ms)	Total count	Avg time (ms)	Avg rows
select book0_.id as id1_2_, book0_.isbn as isbn2_2_, book0_.author as author3_2_, book...	64,7	150	0.43	1,0
select readbook0_.id as id1_11_, readbook0_.book_id as book_id2_11_, readbook0_.user_u...	35,3	60	0.59	1,0
select user0_.username as username1_16_0_, user0_.enabled as enabled2_16_0_, user0_.pa...	22,0	60	0.37	1,0
select genre0_.id as id1_3_, genre0_.name as name2_3_ from genres genre0_ order by gen...	17,3	30	0.58	24,0
select review0_.id as id1_13_, review0_.book_id as book_id5_13_, review0_.opinion as o...	17,0	30	0.57	3,0
select book0_.id as id1_2_0_, book0_.isbn as isbn2_2_0_, book0_.author as author3_2_0_...	16,3	30	0.54	1,0
select wishedbook0_.book_id as col_0_0_ from wished_book wishedbook0_ where wishedbook...	14,6	30	0.49	2,0
select review0_.id as id1_13_, review0_.book_id as book_id5_13_, review0_.opinion as o...	13,6	30	0.45	1,0
select authoritie0_.username as username1_0_, authoritie0_.authority as authorit2_0_ f...	11,2	30	0.37	1,0

Este número tan elevado de queries se debe a las comprobaciones que determina que botones se enseñan, si esto acabara siendo un problema bastaría con mostrar todos los botones y hacer la comprobación cuando sean pulsados.

books/*/publications:

	Total time (ms)	Total count	Avg time (ms)	Avg rows
select publicatio0_.id as id1_10_, publicatio0_.book_id as book_id6_10_, publicatio0_....	108,3	30	3,6	502,0
select book0_.id as id1_2_0_, book0_.isbn as isbn2_2_0_, book0_.author as author3_2_0_...	20,9	30	0.70	1,0
select user0_.username as username1_16_0_, user0_.enabled as enabled2_16_0_, user0_.pa...	13,8	30	0.46	1,0

Vemos que la primera query siendo la que más tarda se trae 502 filas, esto se arreglaría con paginación

Profiling IT API

Después de la realización de las pruebas de rendimiento por los miembros del equipo de desarrollo en diferentes maquinas, con diferentes componentes cada uno, se saca en conclusión que en general el rendimiento de la aplicación es bastante bueno, pero aparte de los escenarios en lo que hay que pasar por el buscador de libros, en donde el rendimiento baja drásticamente, los otros escenarios donde el rendimiento baja pero no tan drásticamente es en los escenarios en los que se pasa o se utiliza el buscador de libros referentes a informática. En estos escenarios se obtienen resultados malos comparados con el resto de los escenarios, pero no tan malos como los obtenidos con los escenarios en los que se pasas por el buscador de libros. Creemos que este problema, al igual que con los escenarios de búsquedas de libro, reside en la API utilizada para realizar las búsquedas de estos libros, a continuación, veremos si esta sospecha es cierta o no.

Vamos a realizar el profiling utilizando el código del Script Load Test de la Historia de Usuario 22: IT API. Esta prueba está compuesta por 2 escenarios:

- Escenario Buscar libro IT: vamos a home, hacemos login, vamos a la sección de IT y buscamos con el parámetro de búsqueda “Java”.
- Escenario Detalle libro IT: Se siguen los mismos pasos y además se entra a ver los detalles de uno de los libros que se obtiene como resultado de la búsqueda.

La prueba se realizaría con 100 usuarios durante 100 segundos en los dos escenarios, dicha cifra de usuarios creemos que es la que más se acerca a la cifra real una vez el sistema esté en producción.

Las siguientes capturas que se adjuntas son del análisis realizado por Glowroot habiendo ejecutado el script de Load Test anteriormente especificado:

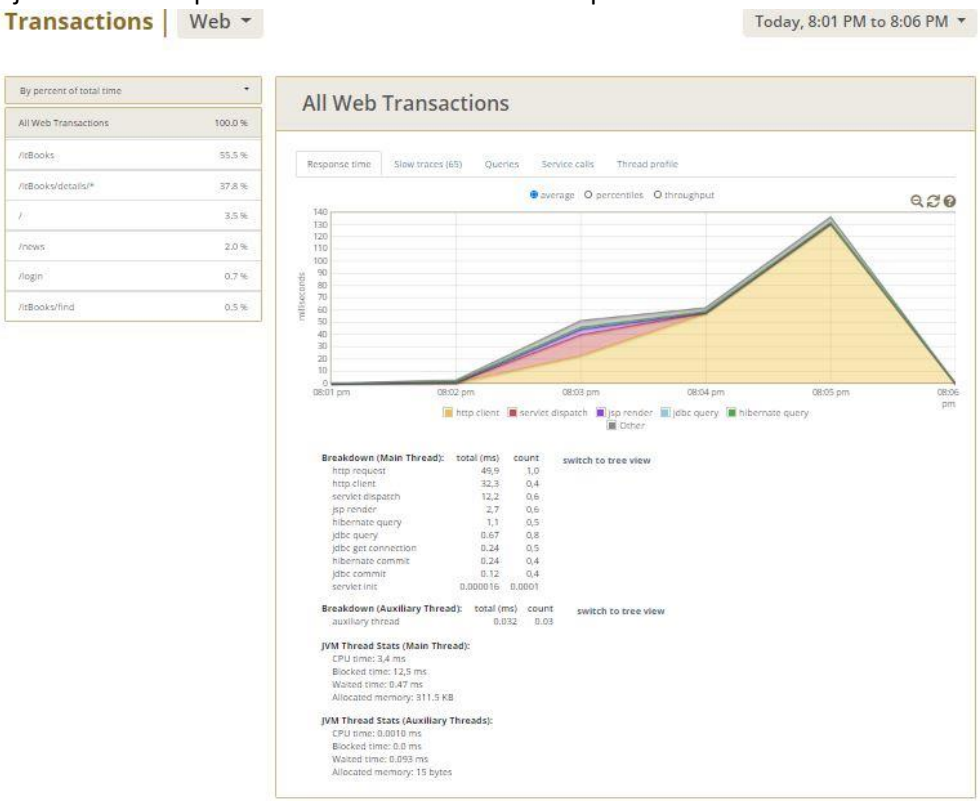
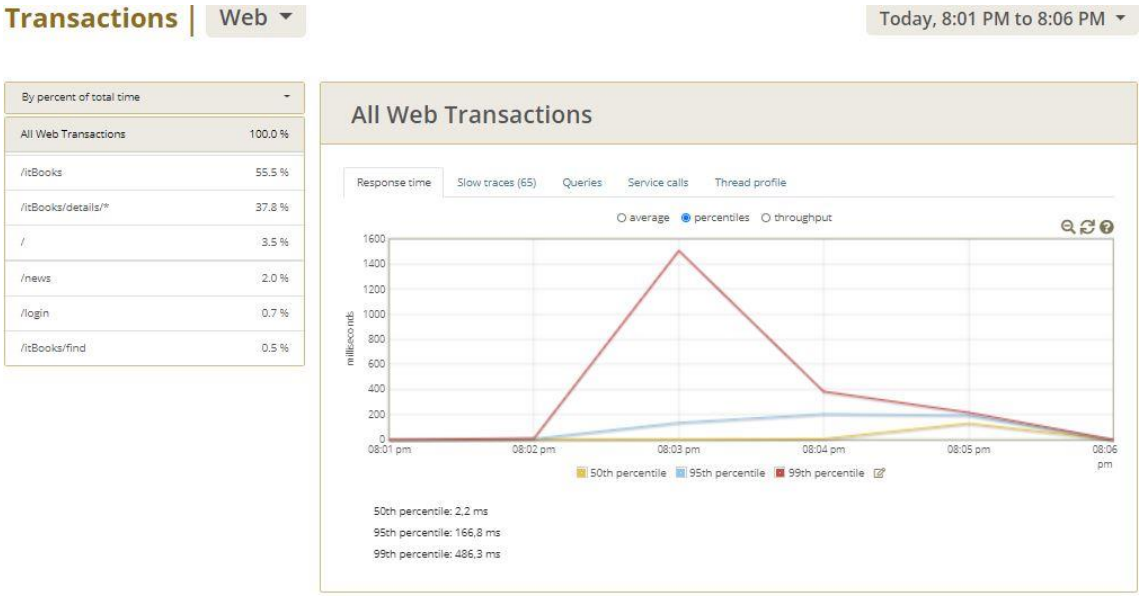
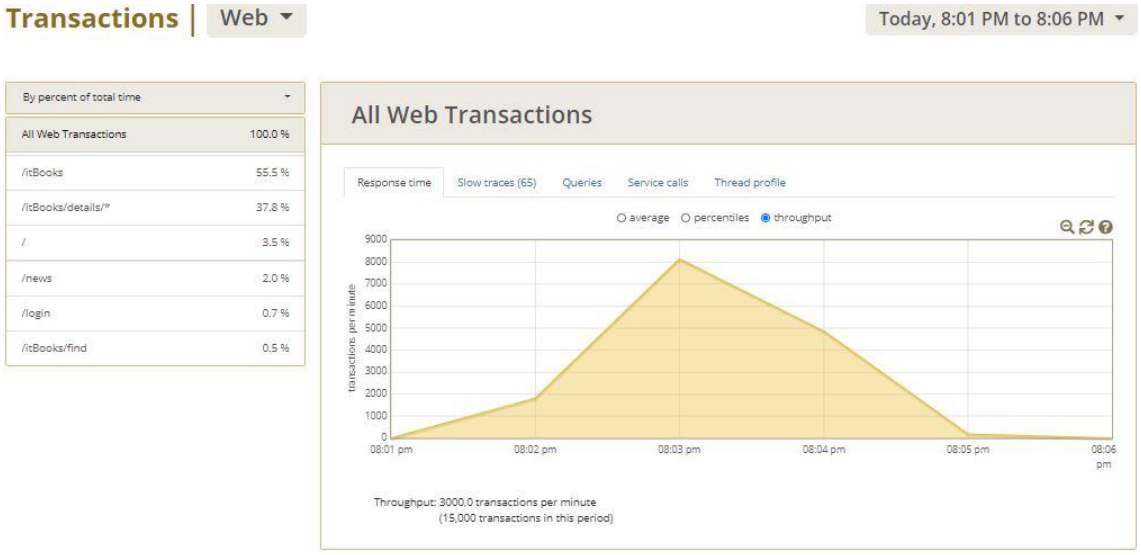


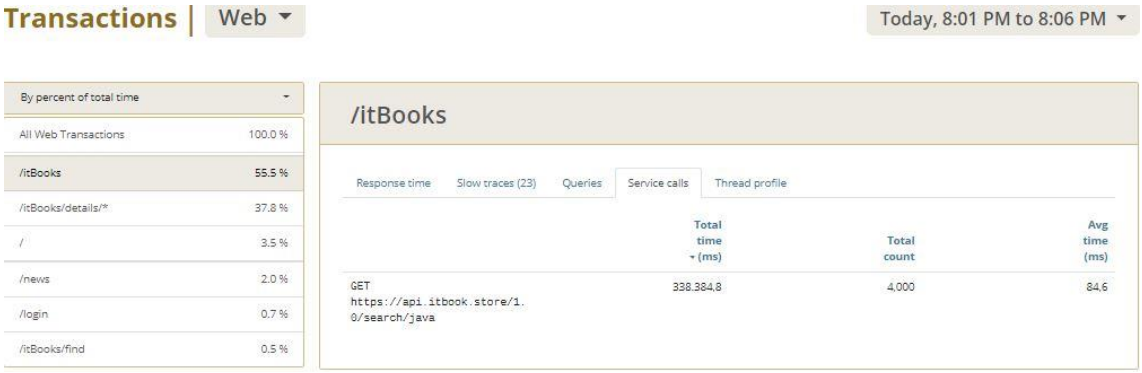
Ilustración 1: All Web Transactions-Average



All Web Transactions-Percentiles



All Web Transactions-Throughputs



Peticiones de servicio

En las capturas proporcionadas se pueden observar los datos referentes a la razón de peticiones por minuto, el porcentaje de peticiones que se han hecho o el tipo de transacciones que se hacen en cada petición, además de los tiempos empleados en ellas y la media de cuantas veces se realiza cada una.

Como se puede observar en la última captura proporcionada podemos observar que las peticiones que se realizan a la API de IT, como vemos se emplea un tiempo total de 338 segundos para resolver 4000 peticiones, lo cual es una cifra demasiado elevada en comparación con el resto de los tiempos obtenidos, pero no tan elevada como la obtenida para la API de poemas.

Después de esto, podemos deducir que efectivamente el cuello de botella se produce en las llamadas a IT API, confirmándose nuestras conjeturas.

La solución a este problema sería realizar una caché en la cual se almacenen los datos, logrando así que no sea necesario llamar a la API tantas veces y devolver con mayor rapidez los resultados.

Refactorización

El objetivo es realizar una refactorización para evitar el cuello de botella que provoca la API https://poemist.github.io/poemist-apidoc.

Para ello, de los procedimientos explicados en la asignatura, nos hemos decantado por hacer uso de una caché. Sin embargo, no es un uso convencional de la misma ya que queremos obtener poemas aleatorios, lo cuál va en contra del concepto de caché.

Para ello, vamos a hacer lo siguiente: en lugar de pedir un poema a la API, vamos a pedir varios (en torno a 10) y guardarlo en caché con un *time out* de 5 minutos. Seremos nosotros los que nos encargaremos de aleatorizar qué poema representar de entre los devueltos en la lista.

Es cierto que el rango de variación de poemas será menor, pero ganaremos mucha eficiencia ya que la API saturaba la aplicación rápidamente.

Implementación

Hasta ahora se hacía una llamada a la API, que devuelve por defecto 5 poemas siempre, y nos quedábamos únicamente con uno, motivo de más para hacer uso de una caché:

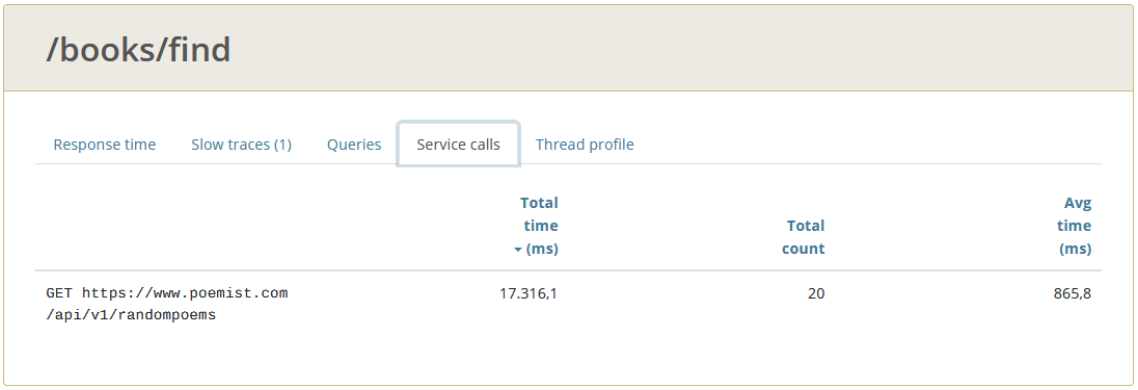
```
public Poem getRandomPoem() {
    final String uri = "https://www.poemist.com/api/v1/randompoems";
    RestTemplate restTemplate = new RestTemplate();
    Poem[] poem = restTemplate.getForObject(uri, Poem[].class);
    return poem[0];
}
```

Ahora, hacemos el método que devuelve la lista cacheable y obtenemos uno aleatorio. Nótese que es necesario crear un atributo *@Autowired* del propio servicio porque de otra forma las llamadas internas no son tenidas en cuenta por la caché:

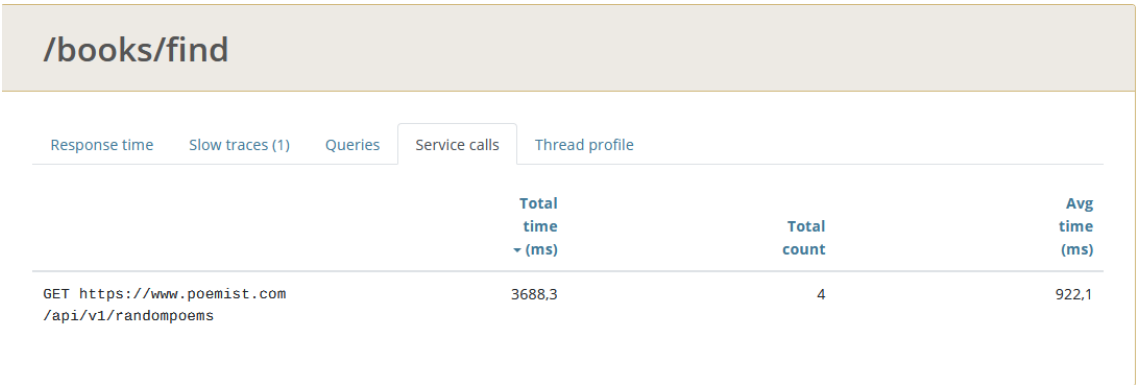
```
@Autowired
private PoemService poemService;

@Cacheable("poemList")
public List<Poem> getPoemsList() {
    List<Poem> poems = new ArrayList<>();
    final String uri = "https://www.poemist.com/api/v1/randompoems";
    RestTemplate restTemplate = new RestTemplate();
    while(poems.size()<10){
        Poem[] poem = restTemplate.getForObject(uri, Poem[].class);
        List<Poem> aux = Arrays.asList(poem);
        poems.addAll(aux);
    }
    return poems;
}

public Poem getRandomPoem() {
    List<Poem> poems = poemService.getPoemsList();
    int random = ThreadLocalRandom.current().nextInt(0,10);
    Poem poem = poems.get(random);
    return poem;
}
```

Resultados

Antes de realizar la refactorización, si entrábamos 10 veces a la página donde se muestra el poema aleatorio, teníamos los siguientes resultados en Glowroot:



Sin embargo, tras implementar la caché, con la primera petición se guardan los resultados (podemos verlo en la consola gracias al logger) y vemos en Glowroot:

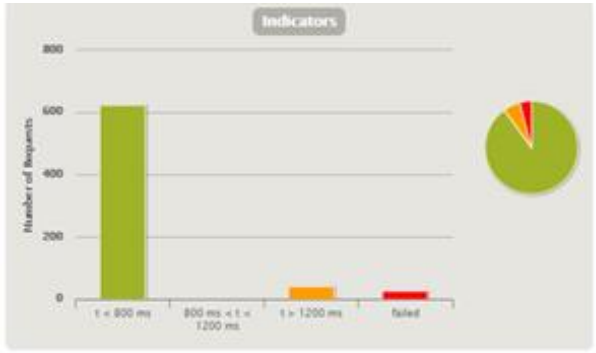


Es cierto que la primera petición es bastante lenta y tadará más de 3 segundos en se respondida, pero a partir de ahí y durante 5 minutos obtendremos un beneficio que compensa esta desventaja ya que no se harán llamadas a la API.

Finalmente, vamos a probar a ejecutar los test de rendimientos de gatling de la HU-03 que llama a la API de nuevo (resultados comparados en la siguiente página).

Podemos apreciar que e¡la primera llamada es muy lenta, tomando 5 segundos y medio, sim embargo el tiempo medio de respesta se reduce en más de la mitad.

Además, la API a partire de las 20 llamadas dejaba de responder por un período de tiempo y así lo evitamos.



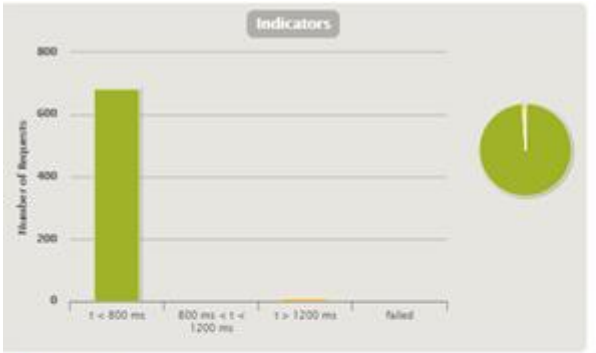
ASSERTIONS

Assertion	Status
Global: max of response time is less than 5000.0	OK
Global: mean of response time is less than 1000.0	OK
Global: percentage of successful events is greater than 95.0	OK

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	693	667	26	4%	3.179	1	14	21	1818	1978	2036	139	455
Home	66	66	0	0%	0.303	5	6	9	11	121	121	10	20
Home Redirect 1	66	66	0	0%	0.303	8	22	29	37	1603	1603	70	271
LoginAdmin	33	33	0	0%	0.151	2	3	4	5	13	17	3	3
LoginReader	33	33	0	0%	0.151	1	3	4	5	13	17	3	3
LoggedReader	33	33	0	0%	0.151	2	4	5	5	24	32	5	5
LoggedAdmin	33	33	0	0%	0.151	2	4	5	5	24	32	5	5
LoggedRe...direct 1	33	33	0	0%	0.151	6	14	16	23	40	59	16	9
FindBooksForm	66	40	26	39%	0.303	175	1822	1991	2123	2493	2506	1260	837
AddBooksForm	33	33	0	0%	0.151	5	11	15	20	320	461	26	77
BookShow	33	33	0	0%	0.151	18	30	36	51	206	277	39	43
AddBook	33	33	0	0%	0.151	6	12	15	19	21	22	12	4
AddBookGoodData	33	33	0	0%	0.151	12	20	23	27	89	89	23	12
AddBookG...direct 1	33	33	0	0%	0.151	10	13	16	20	23	24	14	3
DeleteBookNoAdmin	33	33	0	0%	0.151	8	14	18	21	22	22	15	4
DeleteBookAdmin	33	33	0	0%	0.151	13	15	16	21	39	46	16	6
DeleteBo...direct 1	33	33	0	0%	0.151	9	14	16	21	24	25	14	4

Antes de la caché



ASSERTIONS

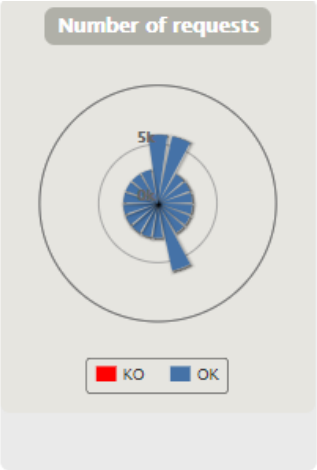
Assertion	Status
Global: max of response time is less than 5000.0	KO
Global: mean of response time is less than 1000.0	OK
Global: percentage of successful events is greater than 95.0	OK

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	693	693	0	0%	3.179	2	13	19	40	2067	5545	89	484
Home	66	66	0	0%	0.303	4	7	9	12	231	232	14	38
Home Redirect 1	66	66	0	0%	0.303	12	22	29	36	2067	2067	85	350
LoginAdmin	33	33	0	0%	0.151	2	3	4	5	22	29	4	5
LoginReader	33	33	0	0%	0.151	2	3	4	5	22	29	4	5
LoggedReader	33	33	0	0%	0.151	3	5	5	6	29	40	6	6
LoggedAdmin	33	33	0	0%	0.151	3	5	5	8	30	40	6	6
LoggedRe...direct 1	33	33	0	0%	0.151	8	14	17	22	41	50	16	7
LoggedAd...direct 1	33	33	0	0%	0.151	7	14	17	27	51	59	16	9
FindBooksForm	66	66	0	0%	0.303	5	10	12	5065	5519	5545	471	1484
BooksList	33	33	0	0%	0.151	10	16	21	26	214	300	26	48
BookShow	33	33	0	0%	0.151	15	28	32	153	315	316	45	66
AddBook	33	33	0	0%	0.151	8	15	18	23	26	26	15	3
AddBookGoodData	33	33	0	0%	0.151	10	20	27	189	287	315	52	70
AddBookG...direct 1	33	33	0	0%	0.151	6	12	14	17	18	18	12	3
DeleteBookNoAdmin	33	33	0	0%	0.151	8	15	22	79	152	152	26	33
DeleteBookAdmin	33	33	0	0%	0.151	12	18	22	186	225	241	44	60
DeleteBo...direct 1	33	33	0	0%	0.151	8	15	17	22	25	26	15	4

Después de la caché

Pero eso era con solo 33 usuarios repartidos en 100 segundos. Con la modificación pasamos a soportar 2900 usuarios en 100 segundos (obviando el KO en el tiempo máximo por la primera petición realizada a la API, que es la que se cachea):



ASSERTIONS

Assertion	Status
Global: max of response time is less than 5000.0	KO
Global: mean of response time is less than 1000.0	OK
Global: percentage of successful events is greater than 95.0	OK

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	60900	60900	0	0%	270.667	0	56	1229	3792	11447	16103	985	1985
Home	5800	5800	0	0%	25.778	1	4	51	8202	11913	12840	1015	2616
Home Redirect 1	5800	5800	0	0%	25.778	2	6	219	1592	2536	13926	359	1020
LoginReader	2900	2900	0	0%	12.889	0	3	1402	8217	11851	12774	1185	2610
LoginAdmin	2900	2900	0	0%	12.889	0	3	1406	8216	11842	12771	1183	2609
LoggedReader	2900	2900	0	0%	12.889	1	8	407	8862	12452	14063	1182	2783
LoggedAdmin	2900	2900	0	0%	12.889	1	8	549	8706	12384	14000	1177	2761
LoggedRe...direct 1	2900	2900	0	0%	12.889	3	17	214	1891	2597	4039	353	658
LoggedAd...direct 1	2900	2900	0	0%	12.889	3	17	206	1926	2782	5220	363	679
FindBooksForm	5800	5800	0	0%	25.778	2	1362	2178	11369	13439	16103	2228	3231
BooksList	2900	2900	0	0%	12.889	5	65	487	1651	2479	5611	372	584
AddBooksForm	2900	2900	0	0%	12.889	2	35	475	1595	2264	5278	352	569
BookShow	2900	2900	0	0%	12.889	8	951	1820	2524	3060	4296	1054	866
AddBook	2900	2900	0	0%	12.889	2	531	1365	2447	2624	4192	769	851
AddBookGoodData	2900	2900	0	0%	12.889	6	400	1162	2247	3231	6613	713	789
AddBookG...direct 1	2900	2900	0	0%	12.889	11	654	1686	2587	3485	5015	966	925
DeleteBookNoAdmin	2900	2900	0	0%	12.889	4	1745	3096	5057	5615	6535	1887	1640
DeleteBookAdmin	2900	2900	0	0%	12.889	5	337	1354	2512	3271	4955	757	913
DeleteBo...direct 1	2900	2900	0	0%	12.889	5	341	1355	2328	3298	4869	745	889

Nuevo test de carga