Gestor de libros usando el módulo de apache mod_proxy_balancer + Pruebas de Carga con Artillery

Daniel Hernández Valderrama , Verónica Osorio Bedoya, Juan José Saavedra Realpe, Jhuly Andrea Vivas Vivas, Santiago Campiño Tamayo

Universidad Autónoma de Occidente

1. Introducción

Este documento expone la propuesta de solución para optimizar el manejo de tráfico en el sistema de gestión de libros de una cadena de librerías mediante balanceadores de carga y pruebas de rendimiento con Artillery, la solución plantea distribuir la carga entre dos servidores backend utilizando el módulo mod_proxy_balancer de Apache, que actúa como punto de acceso único y emplea algoritmos de balanceo para garantizar una respuesta rápida y estable. Además, se realizarán pruebas de carga para evaluar la capacidad y rendimiento del sistema, asegurando un servicio confiable y de alta disponibilidad para los usuarios.

2. Contexto (Problema)

Una cadena de libros busca mejorar la gestión de sus usuarios y llevar un control más eficiente de los libros que adquieren, actualmente el sistema se basa en un servidor que recibe todas las peticiones de libros de los usuarios y las almacena en la base de datos de la librería, sin embargo se ha llegado al punto en que el servidor no tiene la capacidad de gestionar todas las peticiones, lo que ha causado caídas del sistema, perdidas de datos y como resultado insatisfacción del cliente lo que ha afectado su reputación como compañía

Debido a esto se han visto en la necesidad de implementar soluciones que permitan aumentar la capacidad de su sistema asegurando rendimiento, disponibilidad y poder ejecutar planes de contingencia en caso de fallas

3. Alternativas de solución

Para extender la capacidad de gestión del sistema, se debe aumentar la cantidad de servidores, esto permitirá distribuir las peticiones, aumentando la capacidad de gestión y sirviendo como contingencia en caso de que alguno de ellos sufra daños.

Se consideran dos tipos principales de servidores web ampliamente utilizados: **Apache** y **Nginx**.

Apache: es un servidor HTTP de código abierto, conocido por su gran flexibilidad y compatibilidad. Es adecuado para aplicaciones web complejas y escenarios donde se requiere una configuración personalizada, es eficiente en entornos de tráfico bajo a moderado.

Nginx: es un servidor que también actúa como proxy inverso y balanceador de carga. Su diseño

eficiente lo hace ideal para manejar altas cargas de tráfico y conexiones simultáneas

Se debe tener en cuenta que Apache suele usarse para servidores con tráfico bajo o moderado, o entornos que requieran configuraciones personalizadas; por otra parte, Nginx es más utilizado en sitios de alto tráfico o en situaciones donde el balanceo de carga y el proxy inverso son necesarios.

Para el manejo de tráfico se va a trabajar con balanceadores de carga los cuales tienen como función distribuir las peticiones entre los servidores, lo que permite disminuir tiempos de respuesta y monitorización de los servidores

Para el balanceo hay diferentes opciones que pueden utilizarse, tales como:

Balanceo con Nginx: El objetivo principal es mejorar el funcionamiento resistente y eficiente de las aplicaciones web, permite escalabilidad, gestión de certificados SSL/TLS y carga de conexiones simultáneas; es utilizado normalmente en sitios de alta disponibilidad tales como comercio electrónico, redes sociales o aplicaciones de alto volumen de usuarios, se puede utilizar como proxy de Apache para aprovechar la eficiencia en la gestión de conexiones.

Ofrece varios métodos de balanceo de carga, como lo son: Round Robin el cual rota las peticiones de los clientes uniformemente entre todos los servidores, método de conexiones mínimas el cual dirige las nuevas peticiones al servidor con menos conexiones y Hash de IP, en este caso la dirección IP del cliente se utiliza como clave para determinar el servidor que debe gestionar la petición del cliente

Azure Application Gateway: es un equilibrador de carga de tráfico web (OSI capa 7) que permite administrar el tráfico a las aplicaciones web, realiza enrutamiento basado en atributos adicionales de una solicitud HTTP, es utilizado en aplicaciones basadas en microservicios ya que permite enrutar el tráfico a diferentes microservicios en función de la URL

Ofrece varios métodos de balanceo de carga como lo son: Round Robin que distribuye el tráfico de manera equitativa entre los servidores de backend, Cookie-Based Affinity el cual mantiene a los usuarios en el mismo servidor para mantener sesiones y el método de ponderación de rutas la cual permite asignar diferentes pesos a diferentes servidores para manejar cargas desiguales.

Balanceo de carga con Apache: permite distribuir el tráfico entre varios servidores backend o instancias de la aplicación, utilizando diferentes algoritmos de balanceo, puede configurarse tanto en modo proxy de reenvío o inverso, los proxies de reenvío permiten a los clientes acceder a sitios arbitrarios a través de su servidor, ocultando su verdadero origen, mientras que el proxy inverso (o gateway) se presenta ante el cliente como un servidor web normal, sin requerir configuración especial en el cliente.

Para la configuración hay diferentes tipos de módulos de proxy en apache

mod_proxy: proporciona capacidades de proxy básicas, Proxy directo o inverso para reenviar solicitudes a otros servidores, actúa como un módulo de proxy genérico que permite que Apache redirija y reenvíe solicitudes a otros servidores.

mod_proxy_balancer: proporciona balanceo de carga para todos los protocolos soportados, utiliza varios algoritmos de balanceo de carga para distribuir las solicitudes entre múltiples servidores backend.

mod_proxy_hcheck: permite realizar una comprobación dinámica del estado de los miembros del balanceador, ya que permite realizar peticiones periódicas a los servidores backend para asegurar que están en buen estado,

Para asegurar la efectividad de esta implementación, se realizarán pruebas de carga, para esto se tienen diversas herramientas tales como Apache JMeter, Gatling, Apache Benchmark y Artillery, estas herramientas permiten simulan el acceso simultáneo permitiendo evaluar el rendimiento y la capacidad de respuesta del sistema bajo condiciones de alta demanda.

4. Diseño de la solución

Para garantizar la eficiencia de la solución de nuestro, se ha estructurado en las siguientes secciones:

4.1 Servidores web

El sistema contará con 2 servidores web basados en Apache para gestionar las peticiones entrantes

La arquitectura del proyecto web se ha definido de la misma manera para los servidores webServer1 y webServer2, ambos tendrán una vista de usuarios en las que se podrán añadir, listar y una vista para libros en la que se pueden añadir, listar y buscar libros según ID de usuario

4.2 Balanceador de carga

Considerando las diferentes opciones para el balanceo, se ha decidido utilizar los módulos que proporciona apache, se empleará el mod_proxy el cual será configurado en modo inverso, de manera que el balanceador de carga actúe como un único punto de entrada para los usuarios y distribuya las solicitudes a los servidores backend.

Se utilizará el módulo Apache mod_proxy_balancer para la configuración de carga considerando las funcionalidades que tiene, este módulo cuenta con diferentes algoritmos de balanceo, en este caso se trabajará principalmente con mod_lbmethod_byrequests

4.3 Base de datos

Considerando la estructura del sistema se decidió implementar una maquina en la cual se configura la base de datos con la finalidad de que ambos servidores se conecten a la misma base de datos.

4.4 Pruebas de carga

Para validar el rendimiento de la solución, se utilizará Artillery para simular el acceso concurrente de usuarios. Esto permitirá evaluar la capacidad de respuesta y el rendimiento del sistema bajo condiciones de alta demanda

4.5 Arquitectura

A continuación, se presenta la arquitectura final plateada para la implementación

Figura 1. Arquitectura final

5. Implementación

La arquitectura del proyecto web se ha definido de la siguiente manera para los servidores webServer1 y webServer2

├── init.sql : archivo base de datos

├── webapp: carpeta principal

│ ├── books

│ │ ├── controllers ---| book_controller.py

│ │ └── models --- └── book_model.py

│ ├── config.py : se establecen las variables globales para la aplicación

│ ├── db -- └── db.py: gestiona la conexión a la base de datos.

│ ├── run.py : ejecuta el servidor web y define el punto de entrada de la aplicación.

│ ├── users

│ │ ├── controllers -- └── user_controller.py

│ │ └── models --- └── user_model.py

│ └── web: agrupa los archivos de frontend (HTML, CSS, JavaScript).

│ ├── static -- └── script.js

│ └── templates -- ├── books.html ├── editbook.html ├── edit.html ├── edit.html ├── edit.html

└── views.py: configuración de rutas y vistas para la aplicación.

REPOSITORIO:

https://github.com/JJSaavedra52/PFSerTel

A continuación, se especifican IP de cada servidor: WebServer1: 192.168.50.10 y WebServer2: 192.168.50.20

Vista Users

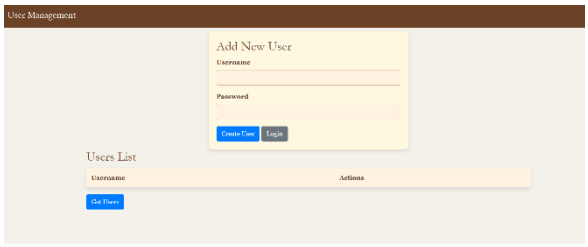

Figura 2. Vista usuarios

Vista Books

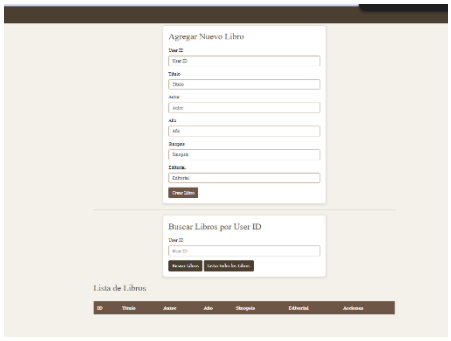

Figura 3. Vista libros

Para la gestión de la base de datos se ha creado la base de datos en una tercera maquina llamada dataBase con el fin de centralizar la gestión de la bd

Para la implementación del balanceador de carga se creó una cuarta maquina con ip 192.168.50.30 la cual tiene una carpeta principal **load-balancer-conf y 2 archivos**

lb-manager.conf

```
<Location "/lb-manager">
        SetHandler      balancer-manager
        allow           from all
        AuthType        "basic"
        AuthName        "lb-manager"
        AuthUserFile    /etc/apache2/htpasswd
        Require         valid-user
</Location>
~
~
```

Figura 4. Archivo de configuración lb-manager.conf

Esta configuración habilita una interfaz de administración de balanceo de carga en Apache, se utiliza el módulo balancer-manager, que permite monitorear y administrar los balanceadores de carga directamente desde un navegador web,

load-balancer.conf

En este archivo crea un balanceador llamado balancer://webapp que se encarga de distribuir las solicitudes entrantes entre dos servidores backend, se utiliza el algoritmo de balanceo byrequests el cual distribuyen equitativamente las solicitudes entre los servidores en función de cuántas solicitudes ha manejado cada uno.

```
<Proxy balancer://webapp>
        BalancerMember  http://192.168.50.10:80
        BalancerMember  http://192.168.50.20:80
        ProxySet        lbmethod=byrequests
</Proxy>
ProxyPass               "/lb-manager"   "!"
ProxyPass               "/"             "balancer://webapp/"
ProxyPassReverse        "/"             "balancer://webapp/"
~
~
~
```

Figura 5. Archivo de configuración load-balancer.conf

Ruta para acceder al módulo del balanceador en web :http://192.168.50.30//lb-manager

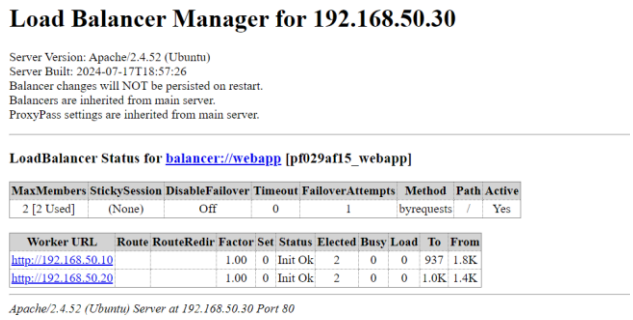

Figura 6. Balanceador de carga web

6. Pruebas

A continuación, se presentan las pruebas para comprobar el funcionamiento básico de la implementación

- Creación usuario #4 y lista de usuarios

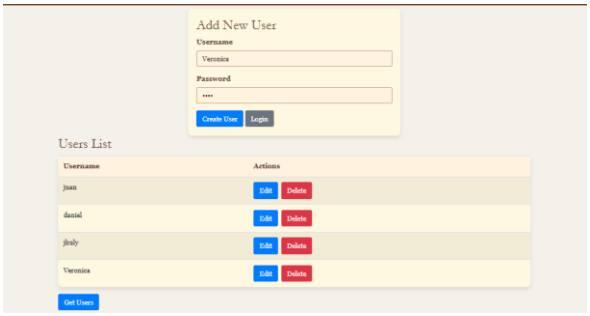

Figura 7. Prueba usuarios

- Creación y búsqueda de libros por ID de usuario

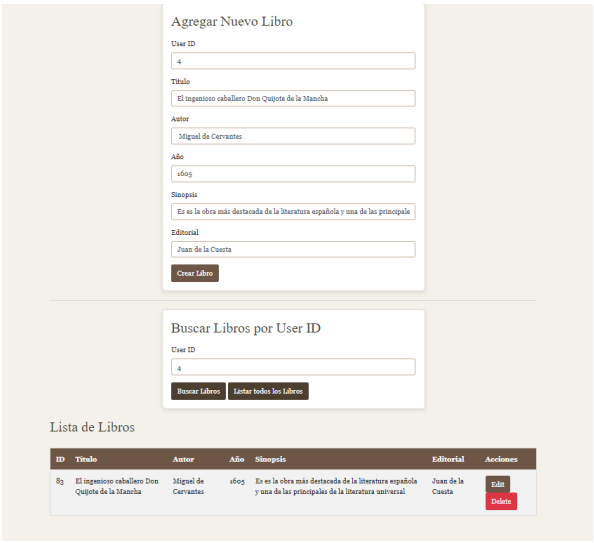

Figura 8. Prueba libros

Pruebas de redirección desde el manager de carga

Load Balancer Manager for 192.168.50.30

Server Version: Apache/2.4.52 (Ubuntu)
Server Built: 2024-07-17T18:57:26
Balancer changes will NOT be persisted on restart.
Balancers are inherited from main server.
ProxyPass settings are inherited from main server.

LoadBalancer Status for balancer://webapp [pf029af15_webapp]

MaxMembers	StickySession	DisableFailover	Timeout	FailoverAttempts	Method	Path	Active
2 [2 Used]	(None)	Off	0	1	byrequests	/	Yes

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	Busy	Load	To	From
http://192.168.50.10			1.00	0	Init Ok	9	0	0	3.8K	32K
http://192.168.50.20			1.00	0	Init Ok	9	0	0	3.9K	4.7K

Apache/2.4.52 (Ubuntu) Server at 192.168.50.30 Port 80

Figura 9. Prueba balanceador 1

Se hizo una petición para listar libros y en este caso fue redireccionado al webServer1

Load Balancer Manager for 192.168.50.30

Server Version: Apache/2.4.52 (Ubuntu)
Server Built: 2024-07-17T18:57:26
Balancer changes will NOT be persisted on restart.
Balancers are inherited from main server.
ProxyPass settings are inherited from main server.

LoadBalancer Status for balancer://webapp [pf029af15_webapp]

MaxMembers	StickySession	DisableFailover	Timeout	FailoverAttempts	Method	Path	Active
2 [2 Used]	(None)	Off	0	1	byrequests	/	Yes

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	Busy	Load	To	From
http://192.168.50.10			1.00	0	Init Ok	10	0	-100	4.3K	46K
http://192.168.50.20			1.00	0	Init Ok	9	0	100	3.9K	4.7K

Figura 10. Prueba balanceador 2

Pruebas con Artillery

Inicialmente se configuró el balanceador de carga para redirigir todas las solicitudes a un solo servidor, esta prueba sirvió como base para establecer un punto de referencia sobre el rendimiento del sistema sin balanceo.

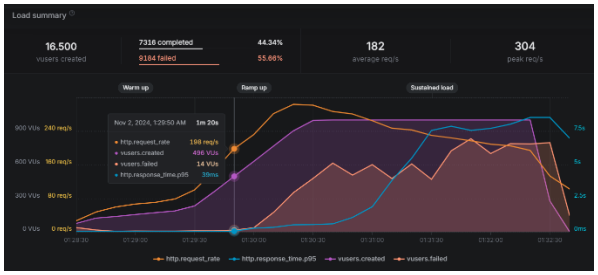

Figura 11. Prueba Artillery solicitudes 1

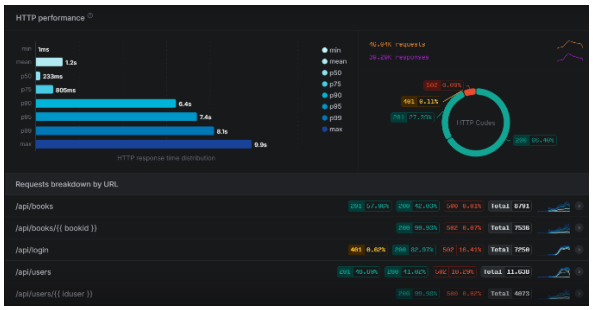

Figura 12. Prueba Artillery métricas 1

Para la siguiente prueba se ha redirigido el balanceador hacia los 2 servidores, esto permitió observar cómo se comportaba el sistema al distribuir la carga entre dos recursos.

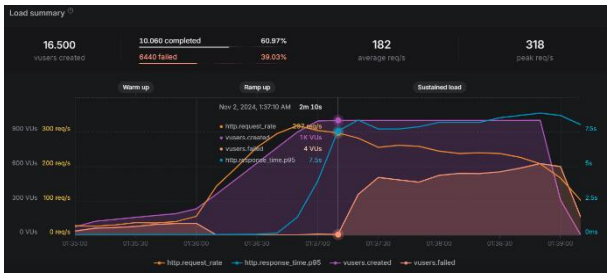

Figura 13. Prueba Artillery solicitudes 2

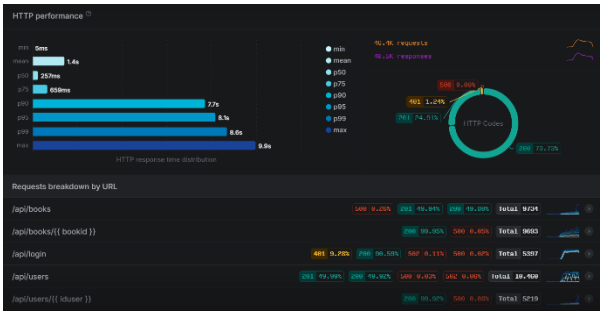

Figura 14. Prueba Artillery métricas 2

También se ha realizado una prueba utilizando el método bybusyness, el cual envía las solicitudes al servidor que está menos ocupado, considerando el número de solicitudes activas que cada servidor está manejando

Figura 15. Prueba Artillery solicitudes 3

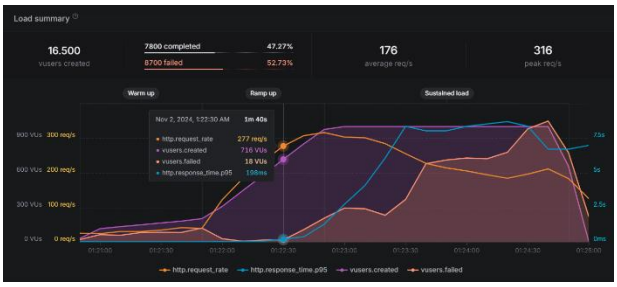

Figura 16. Prueba Artillery métricas 3

7. Discusión de las pruebas

Para la primera prueba se realizaron todas las solicitudes a un solo servidor, resultando en un total de 16,500 solicitudes de las cuales solo el 44% se procesaron exitosamente, este resultado indica que el servidor no tiene la capacidad necesaria para manejar el volumen de tráfico por lo que no es suficiente para garantizar la disponibilidad y el rendimiento del sistema.

Para la segunda prueba, se configuró el balanceador de carga para distribuir las solicitudes entre ambos servidores utilizando el algoritmo **byrequests**, en este caso al enviarse la misma cantidad de peticiones el porcentaje de éxito aumentó al 60%, lo cual demuestra que el balanceador de carga fue capaz de redirigir las solicitudes de manera más eficiente, lo que redujo el número de peticiones perdidas y mejoró la capacidad de respuesta del sistema.

La tercera prueba se llevó a cabo con el objetivo de comparar diferentes algoritmos de balanceo de carga. En este caso, se utilizó el método **bybusyness**, que envía las solicitudes al servidor menos ocupado, evaluando el número de solicitudes activas que cada servidor está manejando en tiempo real, mientras que el método **byrequests**, distribuye las solicitudes de manera equitativa según el número total de solicitudes manejadas por cada servidor desde el inicio

En este caso, dado que ambos servidores tienen capacidades similares, el método **bybusyness** no demostró ser el más efectivo ya que solo el 47% de las solicitudes fueron exitosas, lo que da como resultado que este algoritmo no es el más adecuado para equilibrar cargas en entornos donde los servidores tienen capacidades similares

A continuación, se presenta un cuadro comparativo con los diferentes datos obtenidas en cada prueba

Prueba	Usuarios Completados	Usuarios Fallidos	Requests por Segundo (Promedio/Pico)	Errores de Conexión	Tiempo de Respuesta Promedio (ms)
Un solo servidor	7316	9184	182/304	ECONNRESET (4365), Timeout (2387), Otros (2432)	1.256
Dos servidores	10060	6440	182/318	ECONNRESET (32), Timeout (5867), Otros (541)	1.427
Dos servidores con balanceo "bybussyness"	7800	8700	176/316	ECONNRESET (3024), Timeout (3688), Otros (1988)	1.505

Figura 17. Tabla comparativa resultados

8. Conclusiones

Este proyecto evidencia que, según el escenario planteado, la utilización de un solo servidor no es suficiente para manejar el creciente volumen de solicitudes de los usuarios. Esto resulta en un alto porcentaje de peticiones perdidas, lo que a su vez genera una experiencia insatisfactoria para el cliente.

Se evidencia que el uso del módulo **mod_proxy_balancer** de Apache según sus distintos algoritmos de carga permiten distribuir eficazmente las solicitudes entre múltiples servidores, en este caso el algoritmo **byrequests** mostró resultados positivos, incrementando la tasa de éxito de las solicitudes en comparación con el escenario de un solo servidor; sin embargo, la prueba con el método **bybusyness** demostró que este enfoque es más eficaz en situaciones donde los servidores tienen capacidades desiguales. Esto resalta la importancia de planificar adecuadamente la infraestructura que se va a utilizar, de manera que el método y el algoritmo seleccionados sean los más adecuados según las necesidades específicas del sistema.

En conclusión, la implementación de balanceadores de carga no solo optimiza el manejo del tráfico en el sistema, sino que también garantiza un servicio más confiable y eficiente. Además, nos ha permitido darnos cuenta de la importancia de considerar en qué medida puede aumentar el volumen de usuarios y solicitudes de un producto. De esta manera, se asegura que los servidores estén preparados para manejar la posible escalabilidad del sistema, así como la relevancia de incorporar métodos de balanceo de carga y optimización del rendimiento.

9. Referencias

«Guía de Proxy inverso - Servidor HTTP Apache versión 2.5». https://httpd.apache.org/docs/trunk/es/howto/reverse_proxy.html

«mod_proxy_balancer - Apache HTTP Server Version 2.4». https://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html

«mod_proxy_hcheck - Apache HTTP Server Version 2.4». https://httpd.apache.org/docs/2.4/mod/mod_proxy_hcheck.html

«mod_proxy - Apache HTTP Server Version 2.5». https://httpd.apache.org/docs/trunk/es/mod/mod_proxy.html

Greg-Lindsay, «Características de Azure Application Gateway», *Microsoft Learn*, 21 de junio de 2023. https://learn.microsoft.com/es-es/azure/application-gateway/features

C. Garcia, «Equilibrio de carga en microservicios con NGINX: guía detallada», *AppMaster - Ultimate All-in No-code Platform*,

3 de agosto de 2023. https://appmaster.io/es/blog/equilibrio-de-carga-microservicios-nginx

R. A. Díaz-Heredero. «Tests de rendimiento con Artillery - Adictos al trabajo». Adictos al trabajo. 22 de febrero de 2018. https://adictosaltrabajo.com/2018/02/22/tests-de-rendimiento-con-artillery/