Análisis de balanceo de carga con Nginx

Alejandro C. Fernández, Verónica S. Merlo, Ma. Ángeles Jordán, Leopoldo José Rios[1]

[1] Fac. de Ciencias Exactas Naturales y Agrimensura, Universidad Nacional del Nordeste, Corrientes

cesar.alejandro.fernandez.7@gmail.com, solangeveroniq@gmail.com, angiejordan97@gmail.com, ljr@comunidad.unne.edu.ar

Resumen. El tráfico de datos que manejan a diario los servidores en internet, está en constante crecimiento, lo que puede causar el congestionamiento en el acceso a la información.

En un contexto donde cada día se suman miles y miles de usuarios, es importante entonces, permitir la escalabilidad, disponibilidad y estabilidad de los sistemas, mediante la utilización de algoritmos que faciliten la administración de los servidores, para que las solicitudes entrantes puedan ser atendidas aún en condiciones de alta demanda. En el marco del trabajo de investigación en equipo de la asignatura Redes de Datos, cuarto año de la Carrera LSI, hacemos empleo de Nginx, un software de servidor web de código abierto que permite la utilización de distintos algoritmos de balanceo de carga, para compararlos con un servidor sin balanceo y observar cuál de estos resulta más eficiente para la atención de las peticiones.

Palabras clave: Balanceo de cargas, alta disponibilidad, software libre, cluster.

Abstract. The data traffic handled by servers on the Internet is constantly growing, which can lead to congestion in accessing information. In a context where thousands of users are added every day, it is important to ensure the scalability, availability, and stability of systems by using algorithms that facilitate server management. This allows incoming requests to be handled even under high-demand conditions. In the context of a team research project for the "Redes de Datos" course, in the fourth year of the LSI degree program, we employ Nginx, an open-source web server software that enables the use of different load balancing algorithms. We compare these algorithms with a server without load balancing to determine which one is more efficient in handling requests.

Keywords: Load balancing, high availability, open-source software, cluster.

1 Introducción

En los últimos años, ha habido un crecimiento exponencial en el tráfico en línea. Los usuarios de Internet están mostrando una creciente demanda con respecto a la velocidad de acceso y seguridad, lo que implica que los servidores web deben hacer frente a una carga mayor.

Según el reporte de Cisco acerca del acceso a internet y rendimiento de la red, para el año 2023 habrá 5.300 millones de usuarios de internet en todo el mundo, 3.6 dispositivos y conexiones per cápita y la velocidad promedio de banda ancha fija, será de 110 Mbps. [1] La tecnología de balanceo de carga es una estrategia que permite distribuir de manera equilibrada la carga del sistema, brindando una solución a esta realidad. En otras palabras, esta tecnología posibilita que un grupo de servidores pueda hacer frente a los picos de tráfico y también proporciona una solución de respaldo en caso de fallos. Por ejemplo, si uno de los servidores deja de funcionar, el balanceador redirige el tráfico hacia los servidores restantes que están en línea. El balanceador de carga se encarga de distribuir la carga de trabajo entre los distintos servidores, manteniendo así su capacidad óptima. Esta práctica evita la pérdida de datos debido a una sobrecarga y garantiza un uso adecuado de los recursos del sistema.[2]

Cuando se agrega un nuevo servidor al grupo de servidores, el balanceador de carga automáticamente comienza a enviarle solicitudes. De este modo, los sitios web alojados en servidores con un balanceador de carga serán menos propensos a ralentizaciones o interrupciones del servicio. Para lograr una escalabilidad rentable del sistema y así hacer frente a estos altos volúmenes, las prácticas informáticas modernas suelen implicar la adición de servidores adicionales. Los balanceadores pueden ser instalados tanto en hardware físico como en entornos virtuales para garantizar un equilibrio óptimo de la carga. [3], (Fig.1).

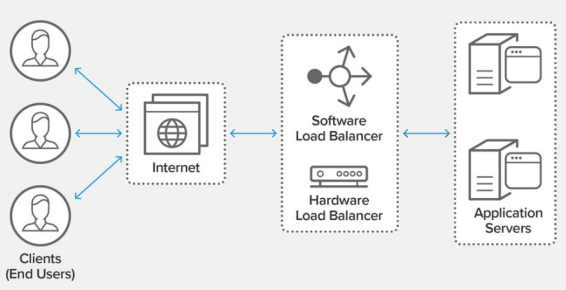

Fig.1 Balanceador de carga

Definiciones:

Balanceo de cargas: El equilibrio de carga se refiere a la distribución eficiente del tráfico de red entrante a través de un grupo de servidores back-end, también conocido como granja de servidores o grupo de servidores. [4]

Servidor: “Los servidores son equipos informáticos que brindan un servicio en la red; dando información a otros servidores y a los usuarios.” Un servidor es un conjunto de computadoras, diseñado para responder a las solicitudes de los clientes y ofrecerles los recursos o servicios solicitados de manera eficiente y confiable. Pueden ser utilizados para alojar sitios web, almacenar y compartir archivos, gestionar bases de datos, ofrecer servicios de correo electrónico, realizar tareas de procesamiento de datos, entre muchas otras funciones.

Los servidores pueden ser ejecutados en diversos tipos de computadoras, incluyendo aquellas que se destinan exclusivamente a esa función y que son conocidas como "servidores dedicados".[5]

Tráfico web: El tráfico web se refiere a la cantidad de datos que se intercambian entre un sitio web y los usuarios que lo visitan. Puede utilizarse para cuantificar la cantidad de visitantes, las páginas vistas y las interacciones que ocurren en un sitio web. Se puede medir utilizando herramientas de análisis web que registran y analizan las solicitudes de páginas web, el tiempo de carga, las fuentes de tráfico, las palabras clave utilizadas para acceder al sitio y otros datos relacionados. El tráfico web es una gran proporción del tráfico de internet.[6]

Los paquetes de software utilizados en esta propuesta son de naturaleza de código abierto -OpenSource-[7], y lo integran los siguientes productos:

- de sistema operativo Linux
- de manejo de containers Docker,
- de balanceo de cargas Nginx,
- de programación de interfaces web,
- de pruebas Apache Benchmark.

La actividad principal consiste en la prueba y análisis de un servidor sin balanceo y otro con balanceo de cargas, teniendo en cuenta los distintos algoritmos disponibles en Nginx.

2 Instrumentación

Para el despliegue del estudio en cuestión se dispuso de los siguientes paquetes de software y hardware:

- Notebook con microprocesador de dos núcleos y 8Gb de memoria RAM
- Sistema operativo Linux en su distribución Ubuntu 22.04
- Docker en su versión 20.10.20 como manejador de contenedores
- Apache Benchmark (ab)
- Nginx en su versión 1.23.2

La distribución de Linux es a elección del usuario. Es esencial para la utilización de Docker, puesto que este necesita para su funcionamiento el kernel de Linux. En sistemas Windows, puede utilizarse el subsistema de Windows para Linux.

Apache benchmark por su parte, es una herramienta para testear el rendimiento de servidores web.

3 Desarrollo

En la implementación se utilizó Docker para crear en contenedores los distintos servidores. Las pruebas realizadas consisten en el envío de solicitudes mediante el uso de la herramienta ab. Se tuvieron en cuenta los casos de un servidor sin balanceo y otro con balanceo probando los distintos algoritmos ofrecidos por Nginx. En ambos casos las pruebas se hicieron para un nivel de concurrencia de entre diez, veinte y cincuenta mil solicitudes.

La instrucción utilizada es la siguiente:

ab -c {clientes} -n {peticiones} http://{dirección}:{puerto}/

Donde clientes es el número de clientes que realizan peticiones de forma concurrente, peticiones es el número de peticiones por cliente, dirección es la IP del servidor y puerto es el puerto por el que escucha el mismo. En el caso de las pruebas con balanceo habrá tres servidores dedicados al servicio html y uno al balanceador de cargas con Nginx.

Es especialmente útil la funcionalidad Docker Compose para que los contenedores puedan desplegarse juntos, puesto que "compose" es una herramienta para definir y ejecutar aplicaciones de Docker de varios contenedores. Se usa un archivo YAML para configurar los servicios de la aplicación. Luego, con un solo comando, se crean y se inicializan los servicios de la configuración donde se alojarán los servidores.[8]

El archivo YAML, para las pruebas con el *balanceador* se verá de la siguiente manera:

version: "3"
networks:
redes:
name: redes
ipam:
config:
- subnet: 172.18.0.0/16
services:
server01:
image: "nginx"
container_name: server01
restart: always
ports:
- 8000:80
volumes:
- "./index.server01.html:/usr/share/nginx/html/index.html"
networks:
redes:
ipv4_address: 172.18.0.4
server02:
image: "nginx"
container_name: server02
restart: always
ports:
- 8001:80
volumes:
- "./index.server02.html:/usr/share/nginx/html/index.html"
networks:
redes:
ipv4_address: 172.18.0.5
server03:
image: "nginx"

container_name: server03
restart: always
ports:
- 8002:80
volumes:
- "./index.server03.html:/usr/share/nginx/html/index.html"
networks:
redes:
ipv4_address: 172.18.0.6
balancer:
image: "nginx"
container_name: balancer
restart: always
ports:
- 8003:80
volumes:
- "./default.conf:/etc/nginx/conf.d/default.conf"
networks:
redes:
ipv4_address: 172.18.0.3

Se genera también el archivo de configuraciones especificado en líneas más arriba “default.conf” donde se detallan las especificaciones para el balanceo con cada algoritmo.

Sin Balanceo: Un solo servidor web recibe todas las peticiones, es uno de los casos más simples.

Round Robin: El algoritmo Round Robin es el predeterminado en los algoritmos de balanceo y el más simple. Distribuye las cargas a cada uno de los servidores activos una por una. [9] (Fig.2)

```
upstream loadbalancer {
   server app1.example.com;
   server app2.example.com;
   server app3.example.com;
}
server {
   listen 80;
   location / {
      proxy_pass http://loadbalancer;
   }
}
```

Fig.2 Configuración del Algoritmo Round Robin

En Nginx “Upstream” se refiere a un grupo de servidores, que normalmente son asignados a un balanceador, “load balancer”. Cada uno de los servidores, se especificarán mediante la palabra “server”. Se puede ver que se ha definido el upstream al mismo nivel con el contexto del servidor. El servidor estará escuchando

en el puerto 80 y reenviará todas las solicitudes al loadbalancer. De forma predeterminada, cada solicitud nueva se enviará mediante proxy al siguiente destino del servidor vecino de forma secuencial rotatoria. Por lo tanto, el tráfico se distribuirá por igual entre los 3 servidores.

También, opcionalmente, podrán cargarse con el parámetro *weight* los pesos, de manera que cuanto mayor sea el peso, más seleccionado será el servidor .[10]

Least Connections: En este algoritmo se debe especificar en las configuraciones "least_conn" envía una nueva solicitud al servidor con la menor cantidad de conexiones actuales con los clientes. La capacidad informática relativa de cada servidor se tiene en cuenta para determinar cuál tiene menos conexiones. (Fig.3)

```
1 upstream loadbalancer {
2    least_conn;
3    server app1.example.com;
4    server app2.example.com;
5    server app3.example.com;
6 }
```

Fig. 3 Configuración del algoritmo Least Connections

Algoritmo Hash*:* El nombre de la directiva es hash y tiene un parámetro opcional llamado consistent. Se distribuyen las solicitudes en función de una clave que define, como la dirección IP del cliente o la URL de la solicitud. Cuando necesitamos más control sobre dónde se envían las solicitudes o queremos averiguar qué servidor upstream es más probable que tenga los datos almacenados en caché, este enfoque es útil. Si agregamos o eliminamos un servidor del grupo, las solicitudes hash se redistribuyen. Para manejar tal situación, el parámetro *consisten*t se puede utilizar para minimizar el efecto de la redistribución. (Fig.4)

```
1 upstream loadbalancer {
2    hash $request_uri consistent;
3    server app1.example.com;
4    server app2.example.com;
5    server app3.example.com;
6 }
```

Fig. 4 Configuración del algoritmo Hash

IP Hash*:* Este algoritmo es ligeramente diferente del algoritmo Generic Hash porque en lugar de usar variables remotas, IP Hash usa la dirección IP del cliente para calcular el valor del hash. Se calculará utilizando los primeros tres octetos de la dirección IPv4 o la dirección IPv6 completa. Este algoritmo garantiza que las

solicitudes de un cliente específico se envíen al mismo servidor upstream durante la sesión del cliente, siempre que ese servidor esté disponible. Por lo tanto, podemos utilizar este algoritmo en aplicaciones stateful, donde el estado de la sesión es importante y no está almacenado en una memoria compartida. También podemos tener en cuenta el parámetro weight con el algoritmo IP Hash (Fig.5).

```
upstream loadbalancer {
   ip_hash;
   server app1.example.com;
   server app2.example.com;
   server app3.example.com;
}
```

Fig. 5 Configuración del algoritmo IP Hash

Random*:* Distribuye de forma alternada la carga entre los servidores disponibles.[9] Este algoritmo seleccionará un servidor del upstream de forma aleatoria y le enviará las solicitudes del cliente. El nombre de la directiva es random. Tiene un parámetro opcional llamado two y le indicará a Nginx que elija aleatoriamente 2 servidores del grupo. Luego, la carga se equilibrará entre esos 2 servidores de acuerdo con el método que especificamos. En este caso, es least_time, pero el método predeterminado es least_conn. También podemos usar el parámetro weight con este algoritmo. [10] (Fig.6).

```
upstream loadbalancer {
   random two least_time=last_byte;
   server app1.example.com;
   server app2.example.com;
   server app3.example.com;
}
```

Fig.6 Configuración del algoritmo Random

4 Comparativa

Se debe considerar que en el entorno de pruebas se comparten los recursos del host anfitrión al utilizarse contenedores, por lo que hay muchas diferencias y limitaciones con respecto a un entorno real como, por ejemplo; que cada servidor posee sus características propias, el tiempo de transferencia también es mayor dependiendo de la velocidad de la red y la cantidad de nodos que el paquete debe atravesar hasta llegar al destino, la cantidad de usuarios y los recursos a los que acceden, entre otras cosas.

Se realizaron múltiples pruebas con el fin de comparar los algoritmos, para diez mil, veinte mil, cincuenta mil, cien mil usuarios y con distintos niveles de concurrencia y características en los hosts de prueba, que en general arrojan resultados variantes de acuerdo al entorno. Si bien contrario a lo que parecía en principio, los valores del servidor sin balanceo son parecidos en cuanto al uso de recursos, e incluso mejores que en los casos con balanceo (teniendo en cuenta las métricas), el uso de la cpu aumenta considerablemente hasta un punto en que el servidor sin balanceo no puede satisfacer todas las solicitudes, volviéndose necesario el balanceo de cargas para asegurar la escalabilidad del sistema. Para Round Robin de 50 mil solicitudes ver (fig.7). Sin Balanceo para 50 mil solicitudes ver (fig.8).

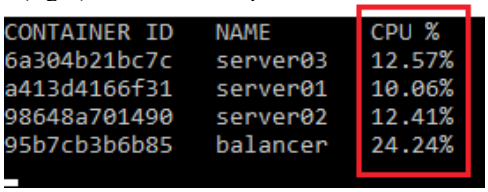

Fig. 7 Round Robin 50 mil solicitudes

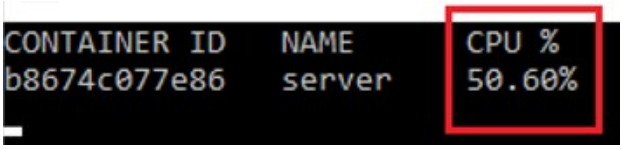

Fig. 8 Sin balanceo 50 mil solicitudes

Este punto varía según las características propias del servidor, por lo que se debe evaluar en qué punto un sistema necesita la utilización de un servidor para balanceo de cargas.

Tabla 1. Comparativa de los servidores

Parámetros	Algoritmos			
	Sin Balanceo	Round Robin	Least Connections	IP Hash
Solicitudes	10000	10000	10000	10000
Uso de la CPU	Servidor 1: 40.71%	Servidor 1: 5.04% Servidor 2: 5.03% Servidor 3: 5.43% Balancer:10.99%	Servidor 1: 11.07% Servidor 2: 1196% Servidor 3: 10.91% Balancer: 26.08%	Servidor 1: 30.46% Servidor 2: 0% Servidor 3: 0% Balancer: 32.86%
Peticiones por segundo	189.47	168.14	124.31	125.98

Ratio de transferencia (Kbytes/seg)	94.55	82.92	61.30	62.13
Tiempo de conexión (ms)	Min.: 3 Máx.: 228 Media: 10 Desvío: 12.9	Min.:3 Máx.:289 Media:12 Desvío:18.2	Min.:2 Máx.:414 Media:9 Desvío:10.6	Min.:2 Máx.:1035 Media:10 Desvío:21.9
Tiempo de procesamiento (ms)	Min.:12 Máx.:450 Media:43 Desvío:32.9	Min.:13 Máx.:642 Media:48 Desvío:37.9	Min.:13 Máx.:522 Media:72 Desvío:37.3	Min.:13 Máx.:640 Media:70 Desvío:37.3
Tiempo de espera (ms)	Min.:9 Máx.:444 Media:39 Desvío:31.5	Min.:12 Máx.:642 Media:47 Desvío:37.7	Min.:12 Máx.:522 Media:70 Desvío:37.0	Min.:13 Máx.:640 Media:68 Desvío:37.2

Con las pruebas, es posible observar que el uso de la cpu con el algoritmo Round Robin se encuentra distribuido de manera uniforme entre los servidores, de este modo la carga resulta más liviana para cada servidor individualmente. Vemos así mismo que el ratio de transferencia disminuyó un poco, aunque casi no se aprecia diferencia entre el tiempo por solicitud en comparación al servidor sin balanceo. El algoritmo es útil cuando todos los servidores tienen recursos similares y se busca una distribución equitativa de las peticiones entre ellos.

El algoritmo Least Connections resulta beneficioso en situaciones donde los servidores tienen capacidades de procesamiento desiguales o se pretende redirigir las solicitudes hacia los servidores menos ocupados. En la prueba, se ve reflejado que el porcentaje de cpu utilizada es mayor que en el algoritmo Round Robin.

Por otro lado, los algoritmos Hash e IP Hash se emplearán cuando se requiere mantener la persistencia de la sesión o utilizar almacenamiento en caché basado en direcciones IP del cliente. En la evaluación del algoritmo IP Hash, como todas las peticiones se hacen desde un host con la misma ip, estas se redirigen al mismo servidor que atendió primero la solicitud, en este caso, el servidor 1. El ratio de transferencia es mayor y el tiempo de conexión es relativamente bajo por estar los datos de la misma en la memoria caché.

Se puede observar que el servidor sin balanceo obtuvo una mejor performance para el número de solicitudes. Sin embargo, cuando las solicitudes inundan el servidor, se necesita implementar el balanceo de cargas para la incorporación de un nuevo servidor, de modo que las solicitudes se repartan entre ellos. Algo a considerar es que el balanceador también tiene un tiempo extra de procesamiento para la decisión del equilibrio de las cargas.

5 Conclusión y posibles mejoras

En el estudio realizado, se han observado ventajas significativas derivadas del uso de un balanceador de cargas. Su utilización permite la escalabilidad del sistema, mejor tolerancia ante fallos, alta disponibilidad y división de la carga de trabajo. Por lo

tanto, en aplicaciones web que enfrentan un alto volumen de peticiones concurrentes por segundo, la implementación de un balanceador resulta fundamental para garantizar un servicio estable y consistente para los clientes.

En relación a los algoritmos de balanceo de cargas, la elección del algoritmo adecuado va a depender del contexto en el que se encuentran los servidores web, los recursos disponibles y la naturaleza de las peticiones. Sin embargo, el de mejor rendimiento en las condiciones de prueba, es sin dudas, Round Robin.

En el futuro, se podrían implementar mejoras al trabajo mediante la utilización de clúster de servidores distribuidos en máquinas físicas diferentes. Esta configuración permitiría que cada servidor cuente con sus propios recursos, lo que significa una simulación más realista del funcionamiento de un balanceador de cargas.

Otra mejora relevante sería la adopción de herramientas de pruebas en varias máquinas para evaluar mejor los algoritmos de balanceo tales como IP Hash.

6 Referencias

1 www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/index.html
2 https://www.ovhcloud.com/es/public-cloud/what-load-balancing/, último acceso 2022.
3 Miquel Talavera Foix, Carolina Albea-Sanchez. Control de balanceo de carga de un grupo de servidores de red. Valencia, España. pp.478-483. ⟨hal-01068501⟩(2014)
4 https://www.nginx.com/resources/glossary/load-balancing/, último acceso 2022.
5 Marchionni E. Administrador de servidores, 1a edición. Fox Andina, Banfield, Lomas de Zamora: Dradi, Buenos Aires, Argentina. (2011).
6 Fundación Telefónica, Sociedad Digital en España 2017, Madrid, España (2017).
7 https://www.gnu.org/philosophy/free-sw.es.html, último acceso 2022.
8 https://learn.microsoft.com/es-es/azure/cognitive-services/containers/docker-compose-recipe, último acceso 2022.
9 Chunga Zuloeta, J. L., Chuzón Sanchez, W. Tesis: Comparación de Algoritmos de balanceadores de carga utilizando Clúster homogéneo en servidores web, USS, Pimentel. Perú. (2017).
10 https://www.purocodigo.net/articulo/balanceo-de-carga-de-alto-rendimiento-con-nginx, último acceso 2022.