Balanceo de Carga de bases de datos con MySQL y HAProxy

William Banguera , Miguel Sotelo
Facultad de ingeniería, Universidad Autónoma de Occidente
Cali, Valle del Cauca

william.banguera@uao.edu.co
miguel.sotelo@uao.edu.co

Abstract) In contexts of high need for operations and data access, load balancing is essential to ensure the availability and efficiency of databases. This report offers a solution for load balancing MySQL databases through HAProxy, an open source proxy recognized for its effectiveness and reliability. Through a multi-node configuration, HAProxy makes it easy to distribute requests between different MySQL servers, optimizing system resources and enhancing fault tolerance. The creation and implementation of a scalable architecture that supports large amounts of traffic, offering redundancy and reducing response times is reported.

I. INTRODUCCIÓN

La estrategia de balanceo de carga en bases de datos es fundamental en sistemas distribuidos para optimizar el desempeño, la disponibilidad y la resistencia a fallos en contextos con alta demanda de transacciones. Para MySQL, la utilización de un balanceador de carga como HAProxy facilita la distribución eficaz de las peticiones entre varios servidores de la base de datos, optimizando de esta manera el aprovechamiento de los recursos y previniendo atascos. HAProxy, una solución muy empleada en el sector para el balance de carga y el proxy inverso, se fusiona con MySQL para establecer una infraestructura sólida capaz de manejar grandes cantidades de información y varias conexiones al mismo tiempo, disminuyendo los tiempos de respuesta y optimizando la experiencia del usuario final.

A continuación se abordarán los conceptos básicos del balanceo de carga en bases de datos, las ventajas de combinar MySQL con HAProxy, y se detalla un proceso de implementación que facilita la creación de un sistema distribuido y tolerante a fallos.

II. PROBLEMA

Uno de los principales desafíos es manejar eficientemente el tráfico hacia las bases de datos, especialmente cuando la demanda de consultas y transacciones aumenta de forma significativa. Sin una arquitectura adecuada para gestionar este tráfico, un único servidor de base de datos suele convertirse en un cuello de botella, lo que afecta el rendimiento de la aplicación, aumenta los tiempos de respuesta y compromete la experiencia del usuario. Además, la falta de redundancia en un solo servidor de base de datos puede derivar en tiempos de inactividad y pérdidas de datos en caso de fallos, lo que amenaza la continuidad del servicio.

El balanceo de carga en bases de datos con HAProxy y MySQL permite distribuir las solicitudes de manera equitativa entre varios servidores, reduciendo la carga en cada uno y mejorando la capacidad de respuesta general del sistema. Esta configuración también introduce redundancia y tolerancia a fallos al permitir que las operaciones se redirijan a otros servidores disponibles si uno falla, minimizando así el riesgo de interrupciones en el servicio. La implementación de un balanceador de carga no solo optimiza el rendimiento del sistema en escenarios de alta concurrencia, sino que también proporciona una base escalable para el crecimiento futuro del sistema, garantizando una infraestructura de base de datos robusta y fiable.

III. ALTERNATIVAS DE SOLUCIÓN

3.1. Balanceo de carga con docker y contenedores

Esta solución utiliza Docker para desplegar múltiples contenedores de MySQL y HAProxy en un solo servidor o en múltiples servidores conectados en una red virtual. Docker simplifica el despliegue y la administración de estos servicios, permitiendo que cada instancia de MySQL y HAProxy se ejecute en su propio contenedor aislado. Esta configuración facilita la escalabilidad y permite agregar o eliminar instancias según la demanda. Algunas de sus ventajas y desventajas son:

Ventajas:

- Escalabilidad: Es sencillo añadir nuevos contenedores de MySQL a medida que aumentan las necesidades de tráfico.
- Aislamiento: Docker asegura que cada instancia de MySQL y HAProxy esté completamente aislada, lo cual mejora la seguridad y la administración.
- Portabilidad: La configuración puede ser replicada fácilmente en diferentes entornos

.

Desventajas:

- Complejidad en la administración de redes: Configurar redes y persistencia de datos en Docker puede ser desafiante.
- Rendimiento limitado: En entornos de alta demanda, la infraestructura física subyacente podría convertirse en un cuello de botella.

3.2. Balanceo de carga con Vagrant en un entorno Ubuntu

Esta solución utiliza Vagrant para crear múltiples máquinas virtuales Ubuntu, configurando una máquina con HAProxy como balanceador de carga y otras máquinas como servidores MySQL. Esta solución permite simular un entorno de producción y probar el balanceo de carga en un entorno controlado. Algunas de sus ventajas y desventajas son:

Ventajas:

- Ambiente de pruebas realista: Permite simular un entorno de producción en una máquina local o servidor, ideal para pruebas y desarrollo.
- Personalización: Cada máquina virtual puede configurarse individualmente, lo que proporciona flexibilidad en la administración de la infraestructura.
- Persistencia de datos: La persistencia de los datos es más fácil de gestionar en un entorno virtualizado que en contenedores.

Desventajas:

- Consumo de recursos: Las máquinas virtuales suelen requerir más recursos que los contenedores, lo que podría afectar el rendimiento en sistemas con hardware limitado.
- Escalabilidad limitada: Para entornos de alta concurrencia, la solución puede volverse costosa y complicada de administrar.

3.3. Balanceo de carga basado en infraestructura de nube (AWS, Azure o Google Cloud)

Esta solución implica implementar HAProxy y MySQL en un entorno de nube, aprovechando la infraestructura de balanceo de carga y los servicios de base de datos administrados ofrecidos por proveedores como Amazon Web Services (AWS), Microsoft Azure o Google Cloud Platform. HAProxy puede configurarse en instancias de servidor en la nube, mientras que los servicios de base de datos administrados permiten escalar automáticamente en función de la demanda. Algunas de sus ventajas y desventajas son:

Ventajas:

- Alta disponibilidad: La infraestructura en la nube ofrece redundancia automática y respaldo ante fallos, garantizando alta disponibilidad.
- Escalabilidad dinámica: Es fácil añadir o reducir capacidad de forma automática sin intervención manual.
- Gestión simplificada: Los servicios de bases de datos administrados se encargan del mantenimiento, parches y backups, reduciendo la carga administrativa.

Desventajas:

- Costos: Dependiendo del uso, los costos en la nube pueden incrementarse rápidamente.
- Dependencia de terceros: La infraestructura en la nube está sujeta a las políticas y precios de los proveedores.

IV. DISEÑO DE SOLUCIÓN

En esta solución específica se utiliza un entorno de máquinas virtuales con Vagrant y VirtualBox. Este entorno permite simular una infraestructura en la que múltiples servidores de base de datos reciben solicitudes distribuidas a través de un balanceador de carga, logrando así una mayor tolerancia a fallos y un rendimiento más eficiente. A continuación, se describe la arquitectura y configuración:

1. Balanceador de carga HAProxy: Es el encargado de actuar como punto de entrada en las solicitudes de base de datos, distribuyendo las consultas entre los servidores de base de datos db1, db2, y db3 con el algoritmo roundrobin.
   IP privada: 192.168.56.101
2. Servidores de base de datos con MYSQL: Para el diseño de esta solución se tienen 3 máquinas virtuales cada una con una instancia de MYSQL para responder a solicitudes de los clientes.
   Direcciones IP privadas:
   - db1: 192.168.56.102
   - db2: 192.168.56.103
   - db3: 192.168.56.104

V. IMPLEMENTACIÓN

5.1 Instalación y configuración de HAProxy:

En el servidor haproxy, se instaló HAProxy y se configuró el archivo haproxy.cfg para balancear la carga entre los tres servidores de base de datos.

Se definió un frontend que escucha en el puerto 3306, y un backend que incluye los servidores MySQL (db1, db2, y db3).

Se utilizó el algoritmo roundrobin, que permite distribuir las solicitudes entrantes de manera uniforme entre los servidores disponibles, logrando un balance de carga efectivo y evitando la saturación de cualquier nodo individual.

5.2 Configuración de replicación maestro-esclavo MYSQL:

En el servidor db1, se configuró MySQL para que funcione como el nodo maestro. En el archivo de configuración my.cnf, se habilitó la replicación registrando el binario de transacciones (mysql-bin).
db1 genera y registra las transacciones en los archivos de registro binario (mysql-bin.log), que se utilizan para sincronizar los datos con los servidores esclavos.

Servidores Esclavos (db2 y db3):
Los servidores db2 y db3 se configuraron como nodos esclavos. Cada uno se conecta al maestro db1, leyendo su posición de registro binario (mysql-bin), y aplicando las transacciones realizadas en db1 para mantener una copia idéntica de los datos.
La configuración de replicación maestro-esclavo permite que db2 y db3 actúen como réplicas de db1, asegurando que las consultas de lectura sean atendidas sin sobrecargar al maestro.

VI. PRUEBAS

Para verificar el desempeño y la correcta distribución de carga en el balanceador, se realizaron pruebas de escritura y lectura utilizando sysbench, una herramienta popular para evaluar el rendimiento de bases de datos. Se llevaron a cabo tres pruebas con diferentes cantidades de usuarios simultáneos: 4, 8 y 16.

6.1. *Prueba de 4 usuarios simultáneos:*

En la primera prueba, se configuró sysbench para simular 4 usuarios concurrentes enviando consultas al balanceador haproxy.
La carga se distribuyó uniformemente entre los tres servidores (db1, db2, db3), y el sistema respondió de manera eficiente sin tiempos de espera significativos. Se puede visualizar en la siguiente imagen:

```
OLTP test statistics:
    queries performed:
        read:                            17850
        write:                           6375
        other:                           2545
        total:                           26770
    transactions:                        1270   (21.12 per sec.)
    deadlocks:                           5      (0.08 per sec.)
    read/write requests:                 24225  (402.88 per sec.)
    other operations:                    2545   (42.33 per sec.)

Test execution summary:
    total time:                          60.1296s
    total number of events:              1270
    total time taken by event execution: 239.8802
    per-request statistics:
         min:                                  125.75ms
         avg:                                  188.88ms
         max:                                  416.44ms
         approx.  95 percentile:               231.36ms

Threads fairness:
    events (avg/stddev):           317.5000/28.02
    execution time (avg/stddev):   59.9700/0.05
```

(4 users).

Se puede visualizar el reporte de estadísticas de prueba de carga generada por sysbench, para la prueba de 4 usuarios se obtuvieron;

Read: Número de consultas de lectura realizadas: 17.850
Write: Número de consultas de escritura realizadas: 6.375
Total entre lectura y escritura ; 24.225

Por último el tiempo promedio de ejecución por solicitud, fue de 188,88 ms, este dato nos sirve para comparar con respecto a las otras pruebas.

6.2. *Prueba de 8 usuarios simultáneos:*

Se incrementó la concurrencia a 8 usuarios, aumentando la carga sobre el balanceador y los servidores MySQL.

```
OLTP test statistics:
    queries performed:
        read:                            18116
        write:                           6470
        other:                           2579
        total:                           27165
    transactions:                        1285   (21.34 per sec.)
    deadlocks:                           9      (0.15 per sec.)
    read/write requests:                 24586  (408.30 per sec.)
    other operations:                    2579   (42.83 per sec.)

Test execution summary:
    total time:                          60.2156s
    total number of events:              1285
    total time taken by event execution: 480.3869
    per-request statistics:
         min:                                  266.30ms
         avg:                                  373.84ms
         max:                                 1123.54ms
         approx.  95 percentile:               446.84ms

Threads fairness:
    events (avg/stddev):           160.6250/6.34
    execution time (avg/stddev):   60.0484/0.07
```

(8 users).

Read: Número de consultas de lectura realizadas: 18.116
Write: Número de consultas de escritura realizadas: 6.470
Total entre lectura y escritura ; 24.586

En promedio esta prueba tardó 373.84 ms, casi el doble en comparación a la anterior.

6.3. *Prueba de 16 usuarios simultáneos:*

Simuló 16 usuarios concurrentes, lo cual permitió evaluar la capacidad máxima del balanceador y de los servidores replicados.

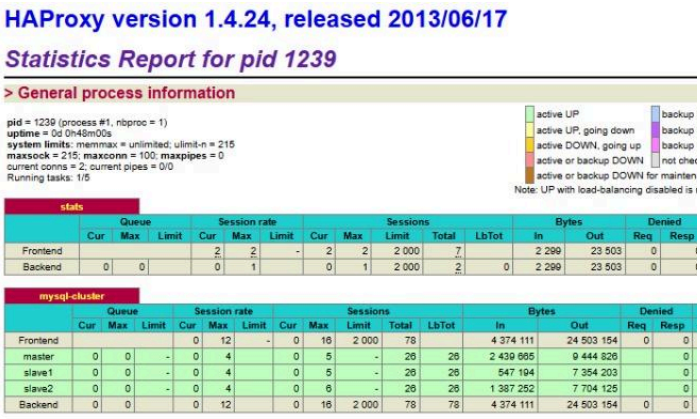
```
OLTP test statistics:
    queries performed:
        read:                            14966
        write:                           5345
        other:                           2085
        total:                           22396
    transactions:                        1016   (16.80 per sec.)
    deadlocks:                           53     (0.88 per sec.)
    read/write requests:                 20311  (335.85 per sec.)
    other operations:                    2085   (34.48 per sec.)

Test execution summary:
    total time:                          60.4761s
    total number of events:              1016
    total time taken by event execution: 964.7802
    per-request statistics:
         min:                                  499.82ms
         avg:                                  949.59ms
         max:                                 6013.82ms
         approx.  95 percentile:              1098.53ms

Threads fairness:
    events (avg/stddev):           63.5000/2.45
    execution time (avg/stddev):   60.2988/0.13
```

(16 users.)

Read: Número de consultas de lectura realizadas: 14.966
Write: Número de consultas de escritura realizadas: 5345
Total entre lectura y escritura ; 20.311

El tiempo promedio de ejecución fue de 949.59ms en comparación al anterior de 8 usuarios.

Por último gracias al servicio HAProxy que nos ofrece un frontend podemos visualizar el número de peticiones por cada servidor:



6.4 Replicación exitosa:

En este apartado se visualiza que las peticiones de lectura se ejecuten correctamente, mostrando por medio del id, que servidor es el que está enviando los datos, como se puede ver a continuación:

```
vagrant@haproxy:~$ mysql -h 127.0.0.1 -u haproxy_root -p'haproxy_root_pass' -D testdb -e "SELECT * FROM test;"
+--------------+
| calificacion |
+--------------+
|            4 |
|            2 |
|           85 |
|           10 |
|           60 |
+--------------+
vagrant@haproxy:~$ sudo mysql -h 127.0.0.1 -u haproxy_root -p'haproxy_root_pass' -e "SHOW VARIABLES LIKE 'serve
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 2     |
+---------------+-------+
vagrant@haproxy:~$ sudo mysql -h 127.0.0.1 -u haproxy_root -p'haproxy_root_pass' -e "SHOW VARIABLES LIKE 'serve
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 3     |
+---------------+-------+
vagrant@haproxy:~$ sudo mysql -h 127.0.0.1 -u haproxy_root -p'haproxy_root_pass' -e "SHOW VARIABLES LIKE 'serve
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 1     |
+---------------+-------+
```

Como era de esperar se puede ver que todos los servidores están funcionando correctamente, ahora veamos que pasa al apagar el servidor db2:

```
vagrant@haproxy:~$ sudo mysql -h
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 1     |
+---------------+-------+
vagrant@haproxy:~$ sudo mysql -h
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 3     |
+---------------+-------+
vagrant@haproxy:~$ sudo mysql -h
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 1     |
+---------------+-------+
vagrant@haproxy:~$ sudo mysql -h
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 3     |
+---------------+-------+
```

Al ejecutar la misma petición se visualiza que ya el servidor 2 no está respondiendo, sin embargo no hay problema ya que la replicación y el balanceo de carga sigue funcionando entre los servidores que están conectados.

VII. DISCUSIÓN DE LAS PRUEBAS

Durante las pruebas de rendimiento realizadas con sysbench, se utilizó un número creciente de usuarios concurrentes (4, 8 y 16) para evaluar la capacidad de respuesta y estabilidad del balanceador de carga configurado con HAProxy y el sistema de replicación Master-Slave en MySQL

Entre los resultados más notorios se puede identificar que con un bajo número de usuarios (4), el tiempo promedio de lectura y escritura fue significativamente menor, lo que indica que el sistema puede manejar cargas ligeras con relativa rapidez. Sin embargo, al aumentar a 8 y luego a 16 usuarios, se observó un incremento en el tiempo promedio por solicitud. Esto es esperado en sistemas OLTP, ya que a medida que aumentan las solicitudes simultáneas, los recursos como CPU, memoria y disco se ven más exigidos, provocando tiempos de espera más largos.

Con el incremento en el número de usuarios, el número total de transacciones y eventos procesados disminuyó. Esto se debe a que el sistema empieza a saturarse y requiere más tiempo para procesar cada solicitud. Aunque el balanceador de carga intenta distribuir las solicitudes de manera equitativa, el aumento de la carga termina afectando la eficiencia general del sistema.

La implementación del algoritmo de balanceo Round Robin en HAProxy demostró distribuir la carga entre los servidores de manera uniforme, contribuyendo a una gestión eficiente de los recursos.

	Bytes		
t	In	Out	R
	4 374 111	24 503 154	
26	2 439 665	9 444 826	
26	547 194	7 354 203	
26	1 387 252	7 704 125	
78	4 374 111	24 503 154	

(Imagen de peticiones por server)

CONCLUSIONES

Este proyecto implementó una solución de balanceo de carga utilizando HAProxy y replicación Master-Slave en MySQL para gestionar un sistema de bases de datos distribuido. La arquitectura diseñada, basada en máquinas virtuales configuradas con Vagrant y VirtualBox, permitió distribuir la carga de trabajo de manera equilibrada entre múltiples servidores de base de datos, aumentando la disponibilidad y tolerancia a fallos. El balanceador de carga configurado con HAProxy y la replicación Master-Slave en MySQL proporcionaron una solución funcional y robusta para la gestión de bases de datos en entornos distribuidos. Este trabajo demuestra cómo la correcta implementación de técnicas de balanceo y replicación puede contribuir a la estabilidad y eficiencia de un sistema de bases de datos, mejorando su capacidad de respuesta ante una carga de trabajo variable y aumentando su disponibilidad.

REFERENCIAS

[1] Díaz-Heredero, R. A. (2012, 19 abril). Balanceando la carga de MySQL con HAProxy - Adictos al trabajo. Adictos Al Trabajo. https://adictosaltrabajo.com/2012/04/19/h-a-proxy-balanceando-carga/

[2] DeBeAndo. (s. f.-b). DeBeAndo - MySQL Balanceo de Carga & Failover con HAProxy. https://debeando.com/mysql-haproxy-balanceo-carga-failover.html

[3] Perfil, V. (2018, 29 marzo). Como replicar una base de MySQL en Linux. https://toquecanela.blogspot.com/2018/03/como-replicar-una-base-de-mysql-en-linux.html

[4] Centro de ayuda de PTC. (s. f.). https://support.ptc.com/help/thingworx/platform/r9/es/index.html#page/ThingWorx/Help/ThingWorxHighAvailability/HAProxyExample.html

[5] MySQL load balancing with HAProxy | Severalnines. (2022, 1 agosto). Severalnines. https://severalnines.com/resources/whitepapers/mysql-load-balancing-with-haproxy/

[6] Tenorio, A. (2022, 6 enero). Balanceando la carga de MySQL con HAProxy - Alexis Tenorio - Medium. Medium. https://alexiseltenorio.medium.com/balanceando-la-carga-de-mysql-con-haproxy-906980f6d30b