Balanceo de Carga de bases de datos con MySQL y NGINX

Alejandro Bravo, Juan Eduardo Jaramillo, Joan Sebastian Balanta, Fernando Jose Cedeño

alejandro.bravo_isa@uao.edu.co
juan_edu.jaramillo@uao.edu.co
joan.balanta@uao.edu.co
fernando.cedeno@uao.edu.co

Servicios Telemáticos
Universidad Autónoma de Occidente
Cali, Colombia

Resumen

Este proyecto implementa y evalúa un balanceador de carga TCP para bases de datos MySQL usando NGINX en modo stream, sobre una topología maestro-esclavo desplegada con Vagrant/VirtualBox. El objetivo es separar lecturas y escrituras: las escrituras van al maestro y las lecturas se distribuyen entre réplicas (esclavos) a través de NGINX, exponiendo puertos distintos para cada tipo de operación. Se preparó un entorno reproducible con cinco VMs (maestro, esclavo 1 y 2, balanceador y cliente) y se ejecutaron pruebas de rendimiento con SysBench. Los resultados muestran incremento de throughput en lecturas al aprovechar réplicas y reducción de la carga directa sobre el maestro, lo que se evidencia como un aumento significativo en el rendimiento, escalabilidad y disponibilidad de los servicios. La solución es práctica para entornos locales y educativos que busquen escalado horizontal de lecturas con bajo coste.

Abstract

This project implements and evaluates a TCP load balancer for MySQL databases using NGINX in stream mode, on a master-slave topology deployed with Vagrant/VirtualBox. The goal is to separate reads and writes: writes go to the master and reads are distributed among replicas (slaves) through NGINX, exposing different ports for each type of operation. A reproducible environment was set up with four VMs (master, slave, balancer, and client) and performance tests were run with SysBench. The results show an increase in read throughput by leveraging replicas and a reduction in the direct load on the master, which is evident as a significant increase in the performance, scalability, and availability of services. The solution is practical for local and educational environments seeking low-cost horizontal scaling of reads.

1. Introducción

El crecimiento acelerado de las aplicaciones web ha evidenciado las debilidades de las arquitecturas monolíticas de bases de datos, especialmente cuando una única instancia de MySQL se encarga de todas las operaciones de lectura y escritura. Este enfoque centralizado alcanza rápidamente sus límites físicos en almacenamiento, procesamiento y memoria, afectando directamente la disponibilidad y el rendimiento del sistema [1][11].

Una estrategia eficaz para superar estas limitaciones es la replicación maestro-esclavo, donde el nodo maestro se encarga exclusivamente de las escrituras, mientras que los esclavos replican los datos y atienden las lecturas [2][5]. Esta separación permite mejorar la escalabilidad, la tolerancia a fallos y el rendimiento general del sistema.

Para aprovechar esta arquitectura, es necesario un mecanismo que distribuya eficientemente las consultas de lectura entre los esclavos. En este contexto, NGINX, tradicionalmente utilizado como servidor web y proxy inverso, puede configurarse como balanceador de carga TCP para MySQL, permitiendo enrutar dinámicamente las conexiones hacia múltiples nodos [3]. Además, su capacidad para realizar chequeos de salud mediante el módulo *ngx_stream_upstream_hc_module* mejora la disponibilidad del sistema ante fallos [4].

El objetivo de este proyecto es diseñar e implementar un balanceador de carga para bases de datos MySQL utilizando NGINX, con el fin de optimizar la gestión de cargas concurrentes de lectura en una infraestructura local. Para validar la efectividad de la solución, se realizaron pruebas de rendimiento y tolerancia a fallos utilizando SysBench [9].

2. Contexto

La creciente demanda de servicios digitales, impulsada por la digitalización de procesos y el aumento del acceso a internet, ha generado presión sobre las infraestructuras tecnológicas, especialmente en la gestión de datos y la disponibilidad del sistema. Según Business Research Insights, el mercado global de desarrollo web alcanzó los 65.350 millones de dólares en 2023 y se proyecta que llegará a 130.900 millones en 2032, con una tasa de crecimiento anual del 8,03 % [10].

Muchas plataformas aún operan bajo arquitecturas monolíticas, donde una única instancia de base de datos gestiona todas las operaciones. Aunque funcionales en etapas iniciales, estas arquitecturas presentan limitaciones críticas al enfrentar cargas concurrentes elevadas, generando cuellos de botella que afectan el rendimiento [11].

Un ejemplo claro es EduConnect, una plataforma educativa utilizada por más de 10 instituciones en Colombia. En el último año, ha experimentado un aumento del 40 % en usuarios activos, especialmente en horarios escolares. Su infraestructura, basada en una única instancia de MySQL, ha comenzado a mostrar signos de saturación, provocando interrupciones frecuentes, tiempos de respuesta prolongados y errores en transacciones críticas.

Este comportamiento ha generado reclamos por parte de las instituciones usuarias y una pérdida de confianza en la plataforma. La situación se agrava al no contar con mecanismos eficientes de tolerancia a fallos ni balanceo de carga que permitan distribuir las operaciones entre múltiples nodos.

Además, estudios como el de Think with Google indican que el 53 % de los usuarios móviles abandonan un sitio web si este tarda más de tres segundos en cargar [12], lo que refuerza la necesidad de adoptar arquitecturas más resilientes y escalables que aseguren la disponibilidad y el rendimiento ante el crecimiento proyectado de la demanda.

3. Alternativas de Solución

Se consideraron varias estrategias para optimizar el rendimiento y la disponibilidad del sistema:

- **Optimización de consultas**: consiste en mejorar los índices, la estructura de datos y eliminar operaciones innecesarias. Esta opción reduce el tiempo de respuesta y la carga del servidor, pero requiere un análisis detallado del uso actual y puede implicar rediseño de la base de datos [13].
- **Escalamiento vertical**: implica aumentar los recursos físicos del servidor maestro (CPU, RAM, almacenamiento). Ofrece una mejora inmediata sin cambios mayores en la arquitectura, pero tiene un costo elevado y está limitado por la capacidad física del hardware.
- **Particionado de tablas**: divide grandes tablas en subconjuntos más manejables, lo que mejora los tiempos de respuesta en consultas específicas. Sin embargo, aumenta la complejidad de mantenimiento y requiere ajustes en las consultas existentes [14].
- **Separación manual de lecturas y escrituras (replicación maestro–esclavo)**: modifica la lógica de la aplicación para enviar operaciones de lectura a réplicas y reservar el maestro para escrituras. Mejora la disponibilidad y la distribución de carga, pero requiere soporte en la aplicación y puede generar lecturas desactualizadas debido al lag de réplica.
- **Replicación maestro-esclavo con NGINX**: la solución seleccionada permite escalar horizontalmente las lecturas de forma transparente, es relativamente simple de desplegar y compatible con la lógica existente en muchas aplicaciones [3][5].

4. Diseño de la Solución

La arquitectura se implementó en cuatro máquinas virtuales, cada una con un rol específico:

- **mysql_master (192.168.70.10)**: nodo maestro que gestiona todas las operaciones de escritura y habilita el registro binario para la replicación [7].
- **mysql_slave1 (192.168.70.11)**: nodo esclavo que mantiene una copia sincronizada de los datos del maestro y atiende consultas de lectura.
- **mysql_slave2 (192.168.70.12)**: nodo esclavo que mantiene una copia sincronizada de los datos del maestro y atiende consultas de lectura.
- **Nginx_balancer (192.168.70.13)**: balanceador TCP que recibe todas las conexiones entrantes y las enruta según el tipo de operación. Se definieron dos canales: uno para lecturas (upstream que incluye al maestro y al esclavo) y otro para escrituras (upstream exclusivo hacia el maestro) [3][4].

- **client (192.168.70.14)**: máquina que ejecuta SysBench para generar carga de trabajo y validar el sistema [9].

Esta arquitectura permite separar eficientemente la carga de trabajo ya que las lecturas se reparten entre el maestro y los esclavos, reduciendo la presión sobre el maestro y mejorando el rendimiento de las consultas. El maestro se dedica exclusivamente a las escrituras, evitando que las operaciones de lectura lo ralentice.

Además, el uso de réplicas de solo lectura permite escalar horizontalmente el sistema simplemente añadiendo nuevos esclavos al grupo de lectura se incrementa la capacidad para consultas *SELECT* sin modificar la lógica de la aplicación [5]. NGINX centraliza el acceso, simplificando la gestión y mejorando la seguridad, ya que los nodos maestro y esclavo no quedan expuestos directamente.

5. Implementación

La implementación se dividió en cinco etapas:

1. **Configuración del maestro**: se habilitó el binlog y se asignó un *server_id = 1* único. Se crearon usuarios con privilegios controlados, incluyendo uno dedicado a la replicación con permisos restringidos [7]. El formato de replicación se estableció como ROW para asegurar la consistencia de los datos. Además, se crearon las bases de datos test y sbtest para las pruebas de carga.
2. **Configuración del esclavo**: Se añadieron dos nodos esclavos, mysql-slave (192.168.70.11) y mysql-slave2 (192.168.70.12), configurados con server_id = 2 y server_id = 3 respectivamente. Ambos servidores se conectan al maestro mediante el comando *CHANGE MASTER TO*, indicando la IP, credenciales y posición en el binlog. Los esclavos aplican los cambios en tiempo real a través del relay log, manteniendo la sincronización con el maestro de forma asíncrona [7].
3. **Configuración de NGINX**: se habilitó el módulo *stream* y se definieron bloques separados para lectura y escritura. Aunque NGINX no interpreta las consultas SQL, la separación por puertos permite enrutar las operaciones correctamente [4][8].
4. **Restricción de acceso**: se aplicaron políticas de IP en MySQL para que solo el balanceador pudiera conectar. Esto refuerza la seguridad y evita la exposición directa de los nodos.
5. **Pruebas con SysBench**: se ejecutaron pruebas de carga simulando escenarios de lectura, escritura y mixtos. Se monitoreó el lag de replicación y se verificó la distribución correcta de las consultas [9].

6. Pruebas

El propósito del plan de pruebas fue verificar el funcionamiento correcto de la arquitectura maestro-esclavo con balanceo de carga mediante NGINX, bajo condiciones controladas que simulan operaciones simultáneas de lectura y escritura. Además de validar la estabilidad del sistema, se buscó medir métricas clave de rendimiento para observar su comportamiento frente a cargas concurrentes.

Entorno de pruebas:

- Cinco máquinas virtuales: una cliente que ejecuta SysBench, una con NGINX como balanceador, y 3 servidores MySQL (maestro y dos esclavos).
- El balanceador redirige las escrituras exclusivamente al maestro (puerto 3308) y distribuye las lecturas entre el maestro y los dos esclavos (puerto 3307).

Metodología:

- Se realizaron pruebas de lectura y escritura, cada una en dos fases.
    - Fase 1: evaluación base con baja concurrencia.
    - Fase 2: incremento progresivo de carga en pasos de 150 hilos hasta saturar el sistema.

Parámetros utilizados:

- Cuatro tablas simuladas con 10,000 registros cada una.
- Ocho hilos concurrentes durante 30 segundos.
- Reportes generados cada 5 segundos para monitoreo continuo.

Monitoreo adicional:

- Se analizaron los registros del balanceador (*mysql_access.log*) para observar la distribución de solicitudes, tiempos de respuesta y errores de conexión.
- También se revisó el estado de replicación con *SHOW SLAVE STATUS* para verificar sincronización entre maestro y los esclavos.

7. Discusión de los Resultados

Durante las pruebas con SysBench se evaluó el comportamiento del sistema bajo distintos niveles de carga y tipos de operación:

- En la fase inicial, con 8 hilos, el sistema mostró un rendimiento estable. Las operaciones de escritura alcanzaron un promedio de 48.82 transacciones por segundo (TPS), con latencias aceptables. Las lecturas, al ser menos complejas, lograron un TPS superior y menor latencia [9].
- Al aumentar la concurrencia en la segunda fase, se observó un incremento significativo en la latencia, especialmente en las escrituras, que superaron los 1400 ms con 150 hilos. Aunque las lecturas mantuvieron un buen TPS, también sufrieron aumentos de latencia conforme se incrementaba el número de hilos.
- A partir de 300 hilos en escritura y 450 en lectura, el sistema alcanzó el límite de conexiones configurado en MySQL (*max_connections*), lo que provocó errores críticos y la interrupción de las pruebas [7].
- Se identificó que, bajo alta carga, la distribución del trabajo entre hilos se vuelve desigual, generando variabilidad en los tiempos de ejecución y afectando la uniformidad del rendimiento.
- La diferencia en latencias entre lectura y escritura confirma que las operaciones de escritura son más costosas en términos de recursos, debido a los mecanismos de bloqueo y consistencia que requieren.

Recomendaciones técnicas:

- Aumentar el parámetro *max_connections* en MySQL.
- Implementar *pools* de conexión para mejorar la gestión de hilos.
- Optimizar índices y revisar las consultas SQL para reducir carga.
- Activar el módulo *ngx_stream_upstream_hc_module* en NGINX para evitar enviar tráfico a réplicas caídas [4].

8. Conclusiones

La solución basada en replicación maestro-esclavo y balanceo de carga con NGINX demostró ser eficaz para mejorar el rendimiento y la disponibilidad en sistemas con alta concurrencia de lectura. La arquitectura permitió distribuir las operaciones de forma eficiente, reduciendo la carga sobre el nodo maestro y mejorando los tiempos de respuesta.

Las pruebas con SysBench confirmaron que esta configuración incrementa la escalabilidad y mejora la tolerancia a fallos ante aumentos en la demanda. Además, se comprobó que es viable en entornos locales con recursos limitados, lo que la convierte en una alternativa accesible para instituciones educativas y organizaciones pequeñas [5][6].

El uso de NGINX como balanceador TCP simplifica la administración del sistema, refuerza la seguridad mediante el aislamiento de los nodos de base de datos, y centraliza el acceso desde un único punto de entrada. En conjunto, esta solución representa una estrategia sólida, flexible y escalable para enfrentar los retos del crecimiento de plataformas digitales.

10. Referencias

[1] D. Arney, "Scaling the Data Layer with MySQL: A Detailed Exploration", Medium. Consultado: el 28 de abril de 2025. [En línea]. Disponible en: https://medium.com/@dinesharney/scaling-the-data-layerwith-mysql-a-detailed-exploration-52c924048399

[2] "Types of Database Replication", GeeksforGeeks. Consultado: el 28 de abril de 2025. [En línea]. Disponible en: https://www.geeksforgeeks.org/types-of-database-replicationsystem-design/

[3] A. Sharif, "Using NGINX as a Database Load Balancer for Galera Cluster", Severalnines. Consultado: el 28 de abril de 2025. [En línea]. Disponible en: https://severalnines.com/blog/using-nginx-database-loadbalancer-galera-cluster/

[4] "Module ngx_stream_upstream_hc_module". Consultado: el 28 de abril de 2025. [En línea]. Disponible en: https://nginx.org/en/docs/stream/ngx_stream_upstream_hc_module.html

[5] GeoPITS, "A Practical Approach to MySQL Replication for Scalability", GeoPITS Blog. Consultado: el 28 de abril de 2025. [En línea]. Disponible en: https://www.geopits.com/blog/a-practical-approach-to-mysqlreplication-for-scalability.html

[6] Severalnines, "HA for MySQL and MariaDB - Comparing Master-Master Replication and Galera Cluster", Severalnines Blog, abril de 2025. Consultado: el 28 de abril de 2025. [En línea]. Disponible en: https://severalnines.com/blog/ha-mysqland-mari

adb-comparing-master-master-replication-galera cluster/

[7] Oracle, “Replication Options for the Binary Log”, MySQL 8.4 Reference Manual. Consultado: el 28 de abril de 2025. [En línea]. Disponible en: https://dev.mysql.com/doc/refman/8.4/en/replication-optionsbinary-log.html

[8] D. Dupont, Complete NGINX Cookbook. Consultado: el 28 de abril de 2025. [En línea]. Disponible en: https://ftp.jaist.ac.jp/pub/sourceforge.jp/c3rb3ru5prjct/69609/Complete_NGINX_Cookbook.pdf

[9] M. A. Lucera, “How to Benchmark Performance of MySQL & MariaDB using SysBench”, Severalnines. Consultado: el 28 de abril de 2025. [En línea]. Disponible en: https://severalnines.com/blog/how-benchmark-performancemysql-mariadb-using-sysbench/

[10] “Tamaño del mercado del desarrollo web, acción - Informe de la industria 2033”. Consultado: el 7 de mayo de 2025. [En línea]. Disponible en: https://www.businessresearchinsights.com/es/marketreports/web-development-market-109039

[11] “Architectural Evolution: From Monolithic Constraints to Microservices Flexibility”, CloudThat Resources. Consultado: el 7 de mayo de 2025. [En línea]. Disponible en: https://www.cloudthat.com/resources/blog/architecturalevolution-from-monolithic-constraints-to-microservicesflexibility/

[12] “Mobile Site Abandonment After Delayed Load Time”, Think with Google. Consultado: el 7 de mayo de 2025. [En línea]. Disponible en: https://www.thinkwithgoogle.com/consumerinsights/consumer-trends/mobile-site-load-time-statistics/

[13] S. Agrawal, V. Narasayya, and B. Yang, "Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design," in Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pp. 359- 370, 2004. Disponible en: https://dl.acm.org/doi/10.1145/1007568.1007609

[14] H. Abadi, A. Marcus, S. Madden, and K. Hollenbach, "Scalable Semantic Web Data Management Using Vertical Partitioning," in Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB), pp. 411-422, 2007. Disponible en: https://api.semanticscholar.org/CorpusID:5581955

[15] Isajar, A. Bravo. “proyecto_telematicos” AlejoBI, "proyecto_telematicos," GitHub. Disponible en: https://github.com/AlejoBI/proyecto_telematicos

ByteForge S.A.S, (2025, 10 de Noviembre). *Load Balancer: nginx + MySQL*. [Video]. Google Drive. telematicos_final.mp4