Balanceador de Carga de Bases de Datos en MySQL y NGINX.

Sergio Herrera, Ludy Astrid, Edwin Guerrero, Sebastián Marinez.

Universidad Autónoma de Occidente
Calle 25 #115-85 Km 2, Cali - Colombia
sergio_and.herrera@uao.edu.co
ludy.agudelo@uao.edu.co
edwin.guerrero@uao.edu.co
sebastian.marinez@uao.edu.co

Resumen

Este proyecto aborda la mejora del rendimiento y disponibilidad en sistemas con alta concurrencia de lectura mediante la implementación de un balanceador de carga para bases de datos MySQL utilizando NGINX. Se configuró una arquitectura maestro-esclavo distribuida en máquinas virtuales, donde el nodo maestro gestiona las escrituras y los nodos esclavos atienden las lecturas. NGINX fue empleado como balanceador de carga TCP para distribuir eficientemente las solicitudes entre los esclavos, integrando chequeos de salud para mayor tolerancia a fallos. Las pruebas se realizaron usando SysBench, evaluando escenarios con distintas cantidades de hilos concurrentes. Los resultados mostraron mejoras significativas en rendimiento, escalabilidad y disponibilidad, reduciendo la sobrecarga en el servidor maestro. Esta solución es aplicable a infraestructuras locales de bajo costo, ofreciendo una alternativa eficiente frente a las limitaciones de arquitecturas monolíticas.

Abstract

This project focuses on improving performance and availability in high-read-concurrency systems by implementing a MySQL database load balancer using NGINX. A master-slave architecture was deployed on virtual machines, with the master node handling write operations and slave nodes serving read requests. NGINX was configured as a TCP-level load balancer to efficiently distribute queries among slaves, incorporating health checks to enhance fault tolerance. Performance tests were conducted using SysBench under various thread loads. The results showed significant improvements in scalability, system availability, and reduced workload on the master node. This solution demonstrates an efficient and low-cost alternative to overcome the limitations of monolithic database architectures.

I. Introducción

Las aplicaciones web en crecimiento hacen evidentes las limitaciones de las arquitecturas monolíticas de bases de datos, particularmente ante incrementos en la demanda de servicios y cargas de trabajo concurrentes. Cuando una única instancia de MySQL centraliza tanto operaciones de lectura como escritura, eventualmente alcanza sus límites físicos en capacidad de almacenamiento, procesamiento y memoria, afectando directamente la disponibilidad y rendimiento del sistema [1].

Una de las estrategias más efectivas para mitigar este problema es la implementación de arquitecturas de replicación maestro-esclavo (master-slave), donde el servidor maestro maneja las operaciones de escritura, y múltiples servidores esclavos replican sus datos para atender operaciones de lectura [2]. Esta separación no solo mejora el rendimiento, sino que también incrementa la tolerancia a fallos y la escalabilidad del sistema [1].

Para aprovechar esta arquitectura, se requiere de un mecanismo eficiente que distribuya las solicitudes de lectura entre los distintos esclavos. En este contexto, NGINX, conocido principalmente como servidor web y proxy inverso, puede ser configurado como balanceador de carga a nivel TCP (Transmission Control Protocol) para MySQL, permitiendo enrutar dinámicamente las conexiones hacia múltiples nodos de base de datos [3]. Su capacidad de realizar chequeos de salud y reencaminar conexiones fallidas refuerza aún más la disponibilidad general del sistema [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 mejorar la gestión de las cargas concurrentes de lectura en una infraestructura local. Además, se realizarán pruebas de rendimiento y de tolerancia a fallos utilizando SysBench para validar la efectividad de la solución propuesta.

II. CONTEXTO

El crecimiento acelerado de las plataformas web ha generado una demanda cada vez mayor de servicios digitales. Esta tendencia, impulsada por la digitalización de procesos y el aumento en el acceso a internet, ha ejercido presión sobre las infraestructuras tecnológicas, especialmente en la gestión de datos y disponibilidad del sistema. Según Business Research Insights (2023), 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]. Este aumento sostenido ha evidenciado las limitaciones de muchas plataformas, que enfrentan caídas, lentitud en la respuesta y fallos de acceso, afectando la experiencia del usuario y la operatividad de las organizaciones.

Una causa común de estos problemas es el uso de arquitecturas monolíticas de bases de datos, donde una única

instancia centralizada gestiona todas las operaciones de lectura y escritura. Aunque funcionales en etapas iniciales, estas arquitecturas presentan limitaciones críticas al enfrentar cargas concurrentes elevadas, provocando cuellos de botella que degradan el rendimiento del sistema [11].

Un ejemplo de esta problemática se observa en EduConnect, una plataforma web de gestión educativa en línea utilizada por más de 10 instituciones en Colombia. Durante el último año, la empresa ha experimentado un incremento del 40% en usuarios activos, especialmente en horarios escolares. Su infraestructura actual, 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 pérdida de confianza en la plataforma. La situación se agrava al carecer de mecanismos eficientes de tolerancia a fallos y balanceo de carga que permitan distribuir las operaciones entre múltiples nodos.

Según un estudio de Google, el 53% de los usuarios móviles abandonan un sitio web si este tarda más de tres segundos en cargar [12], lo que subraya la necesidad de adoptar una arquitectura más resiliente y escalable que asegure la disponibilidad y rendimiento del sistema ante el crecimiento proyectado de la demanda.

III. ALTERNATIVAS DE SOLUCIÓN

Las posibles soluciones consideradas para optimizar el rendimiento y disponibilidad del sistema incluyen:

- Optimización de consultas mediante mejoras en índices, estructura de datos y reducción de operaciones innecesarias.
- Scaling vertical a través de la ampliación de recursos físicos (CPU, RAM, almacenamiento) en el servidor maestro.
- Particionado de tablas para distribuir los datos en subconjuntos más manejables, mejorando tiempos de respuesta en consultas específicas.
- Separación manual de lecturas y escrituras en la capa de aplicación, dirigiendo las lecturas a réplicas y reservando el maestro para operaciones críticas.

Alternativa de Solución	Descripción	Beneficios Principales	Consideraciones / Desventajas
Optimización de consultas	Refinar las consultas SQL mediante ajustes en los índices, reestructuración de datos y eliminación de operaciones redundantes o costosas.	- Reducción del tiempo de respuesta Menor carga en el servidor	- Requiere análisis detallado de uso actual - Puede implicar rediseño de la base
Escalamiento vertical	Aumentar los recursos del servidor maestro (CPU, memoria RAM, almacenamiento) para soportar una mayor carga de trabajo.	- Mejora inmediata del rendimiento Sin cambios mayores en arquitectura	- Costo elevado - Límites físicos de escalabilidad
Particionado de tablas	Dividir grandes tablas en subconjuntos (por rango, lista u otras estrategias) para facilitar el acceso eficiente a datos específicos.	- Consultas más rápidas sobre subconjuntos - Mejor distribución de datos	- Aumenta la complejidad de mantenimiento - Requiere ajustes en consultas existentes
Separación de lectura y escritura	Modificar la lógica de la aplicación para enviar operaciones de lectura a réplicas y mantener el nodo maestro para escrituras y transacciones críticas.	- Mayor disponibilidad - Mejor distribución de carga entre servidores	- Requiere soporte en la aplicación - Riesgo de lecturas desactualizadas (lag de réplica)

Tabla 1. Comparativa de Alternativas de Solución

IV. DISEÑO DE LA SOLUCIÓN

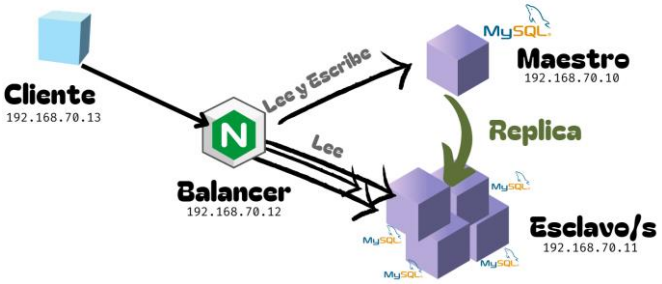

Figura 1. Diseño de la arquitectura

La solución se desplegó en una arquitectura distribuida de cuatro máquinas virtuales, cada una con un rol específico. Los componentes son:

- **mysql_master (192.168.70.10):** nodo maestro de base de datos, que gestiona todas las operaciones de escritura (insertar, actualizar, borrar) y también puede atender lecturas. En él se habilita el registro binario para la replicación.
- **mysql_slave (192.168.70.11):** nodo esclavo de base de datos, que mantiene una copia sincronizada de los datos del maestro. Aplica continuamente los cambios registrados por el maestro y solo atiende consultas de lectura.
- **nginx_balancer (192.168.70.12):** balanceador de carga a nivel de TCP, que recibe todas las conexiones entrantes de los clientes y las enruta según el tipo de operación. Mediante el módulo *stream* de NGINX se definen dos canales: uno para lecturas (upstream que incluye al maestro y al esclavo) y otro para escrituras (upstream exclusivo hacia el maestro). De esta forma, el balanceador distribuye las consultas de lectura entre ambos servidores de base de datos, mientras que las operaciones de escritura van únicamente al maestro.
- **Client (192.168.70.13):** máquina que ejecuta SysBench para generar carga de trabajo y validar el sistema. El cliente se conecta al balanceador (no a los nodos de base de datos directamente), emulando la carga de una aplicación real.

Con esta arquitectura, la carga de trabajo se separa eficientemente: todas las lecturas se reparten entre el maestro y los esclavos, reduciendo la presión sobre el maestro y mejorando el rendimiento de las consultas lecturas [5]. Al estar el maestro dedicado a las escrituras críticas, se evita que operaciones de lectura intensivas lo ralenticen. 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 [6]. El balanceador NGINX centraliza el acceso: los clientes solo conocen un único punto de entrada y no ven la topología interna de la base de datos. Esto simplifica la gestión y mejora la seguridad, ya que los nodos maestro y esclavo no quedan expuestos directamente, sino que aceptan conexiones solo desde el balanceador.

Los beneficios de esta solución son claros: la replicación maestro-esclavo mejora la disponibilidad y la escalabilidad de

las lecturas [5], mientras que NGINX permite aprovechar esas réplicas sin necesidad de lógica adicional en el cliente.

El diseño final entrega en un repositorio de GitHub [15] donde se encontrará el **aprovisionamiento** para poder ejecutar un sistema de base de datos resiliente y eficiente, que balancea automáticamente la carga de lecturas entre varios servidores y concentra todo el acceso a través de un único proxy.

V. IMPLEMENTACIÓN

1. **Configuración del nodo maestro:** Se habilitó el registro binario de MySQL y se asignó un identificador de servidor único (`server_id`) para distinguirlo en la topología [7]. Esto asegura que todos los cambios de datos queden registrados en el binlog, permitiendo que los esclavos los repliquen [7], y que el servidor maestro sea reconocible de forma única. También se crearon cuentas de usuario con privilegios controlados. En particular, se definió un usuario dedicado para la replicación, con solo el privilegio `REPLICATION SLAVE` y restringido a la IP del esclavo. Además, se generaron usuarios de aplicación con permisos mínimos (por ejemplo, solo SELECT o DML) y restringidos al host del balanceador. De este modo, el maestro está listo para servir escrituras y distribuir los eventos de cambio a los esclavos de forma segura.
2. **Configuración del nodo esclavo:** Se configuró el esclavo para que se conectara al maestro e iniciara la replicación. Esto implicó indicarle la dirección IP del maestro, las credenciales del usuario de replicación y el punto de partida en el log binario del maestro. En la práctica, se utilizó la sentencia `CHANGE REPLICATION SOURCE TO` (equivalente a `CHANGE MASTER TO`) para establecer estos parámetros [7]. Con esto, el esclavo comenzó a leer continuamente el binlog del maestro y a aplicar los cambios en su propia copia de la base de datos. Así, el esclavo se mantuvo sincronizado en tiempo real y pudo atender consultas de lectura sin afectar al maestro.
3. **Configuración del balanceador NGINX:** En el servidor de NGINX se habilitó el módulo *stream* para balanceo TCP. Se definieron dos bloques de configuración: uno escuchando en el puerto de lecturas y otro en el de escrituras. El bloque de lecturas encaminó conexiones a un upstream que incluía tanto al maestro como al esclavo, mientras que el bloque de escrituras apuntó únicamente al maestro. De esta forma, NGINX actúa como un proxy inverso a nivel de conexión: distribuye las peticiones entrantes según su canal de origen. Aunque NGINX no discrimina internamente consultas SQL, al disponer de puertos separados se logra enviar de forma transparente las operaciones de lectura al pool compartido y las de escritura solo al maestro [8]. Con esta configuración, se aprovechó la replicación para escalar las lecturas sin afectar la consistencia de las escrituras.
4. **Restricción de acceso a la base de datos:** Para reforzar la seguridad, se aplicaron políticas de acceso basadas en IP en MySQL. Todos los usuarios de la base de datos se definieron con el host establecido, de modo que solo se permitieran conexiones desde el balanceador. Así, únicamente NGINX puede conectar con los puertos 3306 de los nodos maestro y esclavo; cualquier intento directo desde otro origen es rechazado por no existir usuario autorizado. Esta medida garantiza que el balanceador sea el único punto de entrada válido, y que los servidores de MySQL no estén accesibles directamente desde la red externa. La gestión centralizada del acceso simplifica la seguridad y evita exposición innecesaria de los servidores de base de datos.
5. **Pruebas de rendimiento con SysBench:** Finalmente se ejecutaron pruebas de carga desde el cliente para validar todo el sistema. Se utilizó **SysBench**, una herramienta estándar de benchmarks de bases de datos por su simplicidad y soporte de cargas OLTP [9]. Con diferentes escenarios (solo lecturas, solo escrituras y mixto), se midió el rendimiento global y la latencia de replicación. Durante las pruebas se verificó que las consultas de solo lectura se repartían correctamente entre el maestro y el esclavo, y que las operaciones de inserción o actualización llegaban únicamente al maestro, tal como se diseñó. También se monitorizó el *lag* de replicación para asegurarse de que el esclavo seguía al día. Los resultados confirmaron que el throughput aumentaba al agregar réplicas de lectura, y que la arquitectura funcionaba según lo esperado, distribuyendo las cargas como se había planteado.

VI. PLAN DE PRUEBAS

El objetivo principal del plan de pruebas es validar la correcta operación de la arquitectura implementada, basada en un modelo maestro-esclavo de MySQL con balanceo de carga a través de Nginx, bajo condiciones controladas de lectura y escritura simultáneas. A través de estas pruebas, se buscó no solo verificar la estabilidad y funcionalidad del sistema, sino también evaluar métricas clave de rendimiento que permitieran observar el comportamiento del sistema frente a una carga concurrente simulada.

El entorno de pruebas estuvo compuesto por tres máquinas virtuales: una máquina cliente encargada de ejecutar las pruebas de carga mediante `sysbench`, una máquina configurada como balanceador que contenía Nginx y distribuía las solicitudes entrantes, y dos servidores de base de datos, uno actuando como nodo maestro (puerto 3308) y otro como nodo esclavo (puerto 3307). La arquitectura fue diseñada para que el balanceador direccionara las operaciones de escritura exclusivamente al nodo maestro, y las operaciones de lectura hacia el nodo esclavo, aprovechando de esta manera las ventajas de escalabilidad de un sistema replicado.

Se realizaron pruebas de rendimiento divididas en dos tipos: escritura y lectura, cada una con dos fases. La fase 1 evaluó el rendimiento base, mientras que la fase 2 incrementó la carga en 150 hilos por iteración hasta la saturación del sistema. Las

pruebas de escritura, dirigidas al nodo maestro, simularon transacciones concurrentes de inserción de datos, midiendo TPS y latencia. Las pruebas de lectura, en el nodo esclavo, analizaron la capacidad de respuesta ante múltiples hilos simultáneos, evaluando la eficiencia del balanceador de carga y la estabilidad del rendimiento.

Ambas pruebas se ejecutaron con parámetros controlados que buscaban generar una carga significativa pero razonable para el entorno virtualizado. Se definieron cuatro tablas simuladas de 10,000 registros cada una, ejecutadas por ocho hilos concurrentes durante un periodo de 30 segundos. La prueba se repetía en intervalos de 5 segundos para obtener reportes intermedios que permitieran monitorear el comportamiento del sistema a lo largo del tiempo.

Adicionalmente, se monitorearon en tiempo real los logs del balanceador (`mysql_access.log`) con el fin de observar la distribución efectiva de las solicitudes, los tiempos de respuesta y detectar posibles errores de conexión. Esto permitió tener una visión completa tanto desde el punto de vista del rendimiento cuantitativo como del comportamiento operativo del sistema en su conjunto.

VII. DISCUSIÓN DE PRUEBAS

Durante las pruebas de rendimiento realizadas sobre el servidor MySQL utilizando Sysbench, se evaluó el comportamiento bajo distintas cargas y tipos de operaciones, concretamente lectura y escritura, en dos fases diferenciadas por niveles de concurrencia. En la fase inicial, con una baja concurrencia de 8 hilos, el sistema mostró un desempeño estable y consistente. Las operaciones de escritura alcanzaron un promedio de 48.82 transacciones por segundo (TPS) con latencias razonables, mientras que las lecturas presentaron un TPS mayor y latencias aún menores, lo que es coherente con la menor complejidad de estas operaciones.

Al incrementar la concurrencia en la segunda fase, se evidenció un aumento significativo en la latencia, especialmente en las operaciones de escritura, donde el promedio se elevó a más de 1,400 ms con 150 hilos. En lectura, aunque se logró un TPS superior, también se observó un aumento notorio en la latencia conforme se incrementaba el número de hilos. Cabe destacar que, a partir de los 300 hilos para escritura y 450 para lectura, el servidor alcanzó el límite máximo de conexiones configurado en MySQL, provocando errores críticos y la detención de las pruebas. Este comportamiento revela una limitación clara en la configuración por defecto del servidor, que no está preparada para manejar cargas muy elevadas sin ajustes específicos.

Además, se identificó que la equidad en la distribución de trabajo entre los hilos disminuye con la carga, generando variabilidad en los tiempos de ejecución y, en consecuencia, afectando la uniformidad del desempeño. La diferencia significativa en latencias entre lectura y escritura confirma que las operaciones de escritura son más costosas en términos de recursos y tiempo, debido a los mecanismos de consistencia y bloqueo que requieren.

Estos resultados sugieren que, para entornos con alta concurrencia, es indispensable aumentar el parámetro `max_connections` de MySQL y adoptar prácticas de optimización, como el uso de pools de conexión y la revisión exhaustiva de índices y consultas. De esta manera se podría mejorar la escalabilidad y evitar saturación del sistema, garantizando una respuesta más eficiente y estable bajo cargas elevadas. En conclusión, el servidor MySQL demostró ser confiable para cargas moderadas, pero requiere ajustes para operar de forma robusta y eficiente en escenarios de alta concurrencia, evitando así latencias elevadas y fallos críticos que comprometan la estabilidad del servicio.

VIII. CONCLUSIONES

La implementación de un balanceador de carga para bases de datos MySQL utilizando NGINX resultó ser una solución efectiva para optimizar tanto el rendimiento como la disponibilidad del sistema, especialmente en entornos con alta concurrencia de lectura. Esto se debe a que permite distribuir adecuadamente las solicitudes entre múltiples nodos, evitando la sobrecarga del servidor principal.

En este sentido, la arquitectura maestro-esclavo facilitó una distribución eficiente de las operaciones, lo que redujo significativamente la carga sobre el nodo maestro y mejoró los tiempos de respuesta del sistema. Como complemento, las pruebas realizadas con SysBench confirmaron que esta configuración no solo incrementa la escalabilidad, sino que también mejora la tolerancia a fallos ante aumentos en la demanda.

Por otra parte, se comprobó que esta solución es completamente viable en entornos locales con recursos limitados, lo que la convierte en una alternativa accesible y funcional para instituciones educativas y pequeñas organizaciones. Asimismo, el uso de NGINX como balanceador a nivel 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, la solución propuesta representa una estrategia sólida, flexible y escalable para responder a los retos que conlleva el crecimiento de las plataformas digitales.

B. 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-layer-with-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-replication-system-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-load-balancer-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-mysql-replication-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-mysql-and-mariadb-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-options-binary-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-performance-mysql-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/market-reports/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/architectural-evolution-from-monolithic-constraints-to-microservices-flexibility/

[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/consumer-insights/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] E. Guerrero, S.Herrera. “database_balancer” edwingd18, "database_balancer," GitHub. Disponible en: https://github.com/edwingd18/database_balancer.