Evaluación del Enrutamiento Dividido de Lectura/Escritura en una Arquitectura Maestro-Réplica de MySQL Con MySQL Router.

Jean Paul Delgado Jurado, Juan David Muñoz Olave, Karol Lizeth Payares Vizcaino, Maria Camila Orozco Romero
Facultad de Ingeniería y Ciencias Básicas
Universidad Autónoma de Occidente
Cali, Valle del Cauca, Colombia
jean.delgado@uao.edu.co, juan_d.munoz_o@uao.edu.co, karol.payares@uao.edu.co, maria_cam.orozco@uao.edu.co

***Abstract*—This project presents the implementation of a load-balancing solution for MySQL using MySQL Router, combined with a master–replica architecture to improve availability, scalability, and read/write distribution in distributed database environments. The system was deployed on a set of virtual machines configured with Vagrant and VirtualBox, where a master node handles all write operations and replica nodes serve read requests. MySQL Router was used to provide transparent routing, separating read-only and read-write traffic through dedicated logical endpoints. Replication was configured using binary logs to ensure data consistency across nodes.**

Performance tests were conducted with Sysbench to evaluate the system under high-demand scenarios and simulated node failures. Results showed stable operation and correct failover behavior, although the overall performance was strongly limited by the virtualized environment. Latency values remained similar across all scenarios, and no significant evidence was found to indicate an advantage of one routing strategy over another (round-robin vs first-available). Despite these constraints, the solution proved functional and resilient, demonstrating the practical viability of MySQL Router as a load-balancing component in small-scale or development environments.

Keywords*—MySQL Router, load balancing, database replication, master–replica architecture, Sysbench, distributed systems.

I. INTRODUCCIÓN

El aumento constante en la demanda de servicios digitales ha generado la necesidad de garantizar la alta disponibilidad, consistencia de datos y tolerancia a fallos en los sistemas de gestión de bases de datos. En entornos distribuidos, donde múltiples aplicaciones acceden simultáneamente a grandes volúmenes de información, la sobrecarga en un único servidor de base de datos puede degradar el rendimiento general del sistema y comprometer la continuidad del servicio. Frente a este desafío, los mecanismos de balanceo de carga y replicación de bases de datos se han consolidado como estrategias fundamentales para optimizar el acceso concurrente, mejorar la escalabilidad y minimizar los tiempos de inactividad.

En este proyecto se aborda la implementación de un sistema de balanceo de carga para bases de datos MySQL utilizando la herramienta MySQL Router, con el propósito de distribuir eficientemente las operaciones de lectura y escritura entre un conjunto de servidores configurados en un esquema maestro–esclavo. El entorno de desarrollo se compone de múltiples máquinas virtuales configuradas mediante VirtualBox y Vagrant, ejecutando Ubuntu 22.04 como sistema operativo base. El sistema implementado permite asignar las operaciones de escritura al nodo maestro, mientras que las consultas de lectura son delegadas a los nodos esclavos, garantizando así la consistencia de los datos mediante replicación continua.

El objetivo principal de esta implementación es mejorar la eficiencia y la velocidad de acceso a la base de datos, incrementando la tolerancia a fallos y proporcionando una arquitectura escalable y robusta. La solución desarrollada constituye un ejemplo práctico del uso de tecnologías de balanceo y replicación aplicadas a entornos locales de prueba, simulando las condiciones requeridas para infraestructuras de producción con alta demanda transaccional.

II. CONTEXTO DEL PROBLEMA

En entornos donde múltiples clientes o aplicaciones acceden simultáneamente a una misma base de datos, el rendimiento del sistema puede verse comprometido por la concentración de todas las operaciones de lectura y escritura en un único servidor. Este enfoque monolítico genera cuellos de botella que afectan la velocidad de respuesta, incrementan el tiempo de espera en las transacciones y reducen la disponibilidad del servicio en caso de fallos. Además, la imposibilidad de escalar horizontalmente la infraestructura limita la capacidad del sistema para adaptarse a mayores volúmenes de solicitudes o cargas de trabajo dinámicas.

Ante esta problemática, surge la necesidad de diseñar una arquitectura distribuida que permita la distribución equitativa de las solicitudes y la replicación confiable de datos entre distintos nodos. La ausencia de un mecanismo de balanceo adecuado puede ocasionar inconsistencias entre réplicas, pérdida de disponibilidad ante la caída del servidor principal y degradación del desempeño global del sistema. En consecuencia, las organizaciones y desarrolladores requieren soluciones que aseguren la continuidad operativa y la integridad de la información en entornos multiusuario.

El proyecto responde a esta necesidad mediante la implementación de un balanceador de carga para bases de datos

MySQL, configurado sobre un entorno virtualizado de laboratorio. La solución busca mitigar los problemas de saturación, mejorar la respuesta en operaciones concurrentes y garantizar la replicación automática entre servidores maestro y esclavos. De este modo, se plantea una infraestructura con alta disponibilidad y escalabilidad horizontal, que contribuye al desarrollo de competencias en administración avanzada de bases de datos y tecnologías de virtualización orientadas a la tolerancia a fallos.

III. ALTERNATIVAS DE SOLUCIÓN

La problemática de rendimiento y disponibilidad en sistemas de bases de datos distribuidas puede abordarse mediante diversas estrategias tecnológicas. Existen herramientas de balanceo de carga y replicación que permiten optimizar el tráfico de consultas, mejorar la resiliencia del sistema ante fallos y facilitar la escalabilidad horizontal. A continuación, se describen tres alternativas viables para la solución del problema, analizando sus principios de funcionamiento, ventajas, limitaciones y pertinencia en el contexto del presente proyecto.

HaProxy (High Availability Proxy)

HAProxy (High Availability Proxy) es una herramienta de software libre ampliamente utilizada para distribuir tráfico de red entre múltiples servidores, proporcionando alta disponibilidad, balanceo de carga y tolerancia a fallos. Originalmente diseñada para equilibrar solicitudes HTTP, HAProxy ha sido adaptada para trabajar con diversos sistemas, incluido MySQL, convirtiéndose en una opción versátil para gestionar la distribución de operaciones de lectura y escritura en entornos de bases de datos replicadas.

En un entorno MySQL, HAProxy actúa como intermediario entre los clientes y los nodos de base de datos. A través de reglas de configuración, puede dirigir las solicitudes de lectura hacia los servidores esclavos y las operaciones de escritura hacia el servidor maestro. Esta asignación puede realizarse basándose en puertos diferenciados o mediante detección automática de roles, permitiendo que los clientes se conecten a un único punto lógico sin conocer la topología subyacente.

Entre las principales ventajas de HAProxy se destacan:

1. Alta capacidad de personalización de las políticas de balanceo (round robin, least connections, weighted load, etc.).
2. Monitoreo activo del estado de los servidores backend, con conmutación automática (failover) en caso de fallos.
3. Compatibilidad multiplataforma y rendimiento elevado gracias a su arquitectura basada en eventos.
4. Soporte para métricas detalladas, registro de tráfico y administración avanzada mediante interfaz de estadísticas.

Sin embargo, su implementación en entornos MySQL presenta algunas limitaciones. HAProxy no posee un conocimiento interno del estado de replicación de las bases de datos, por lo que requiere scripts externos o herramientas complementarias para distinguir entre nodos maestros y esclavos en tiempo real. Asimismo, carece de integración nativa con el ecosistema MySQL, lo que puede aumentar la complejidad del mantenimiento y la sincronización ante cambios en la topología. [1]

ProxySQL

ProxySQL es un proxy de alto rendimiento diseñado específicamente para entornos MySQL y MariaDB. A diferencia de los balanceadores de carga tradicionales que operan en nivel de red o transporte, ProxySQL trabaja a nivel de aplicación, lo que le permite analizar las consultas SQL y aplicar reglas avanzadas de enrutamiento, caché, priorización y filtrado. Su principal propósito es optimizar el rendimiento y la disponibilidad de los clústeres de bases de datos MySQL mediante una gestión inteligente del tráfico de consultas.

ProxySQL se ubica entre las aplicaciones cliente y los servidores MySQL, manejando múltiples conexiones simultáneas a través de su motor de connection pooling. Este componente reduce la sobrecarga de conexiones directas y mejora significativamente la eficiencia del sistema en entornos de alta concurrencia. Asimismo, el proxy mantiene información sobre el estado de los nodos backend (maestro y esclavos), permitiendo conmutación automática (failover) y reconexión transparente en caso de caída de un nodo. [2]

Entre sus ventajas más relevantes se encuentran:

1. Conocimiento contextual del SQL, lo que permite aplicar reglas de enrutamiento basadas en el tipo de operación (SELECT, INSERT, UPDATE, etc.) o incluso en el contenido de la consulta. [3]
2. Compatibilidad nativa con MySQL y mecanismos de replicación, facilitando la integración con configuraciones maestro–esclavo o clústeres de Galera.
3. Balanceo adaptativo y priorización dinámica de servidores según su carga o latencia.
4. Soporte para caché de consultas, lo que reduce la carga sobre los servidores y mejora los tiempos de respuesta.
5. Herramientas de monitoreo y estadísticas integradas para optimizar el rendimiento.

Así pues, ProxySQL constituye una alternativa sumamente potente para arquitecturas distribuidas de bases de datos que demandan enrutamiento inteligente, escalabilidad y resiliencia. Su capacidad de interpretar consultas SQL lo posiciona como una opción avanzada para entornos de producción de gran escala, aunque su complejidad puede resultar excesiva en implementaciones de pequeña o mediana envergadura.

MySQL Router

MySQL Router es una herramienta oficial desarrollada por Oracle como parte del ecosistema de MySQL InnoDB Cluster. Su función principal es actuar como un middleware de enrutamiento entre las aplicaciones cliente y un conjunto de servidores MySQL configurados en topología maestro–esclavo o en un clúster de alta disponibilidad. A diferencia de soluciones externas como HAProxy o ProxySQL, MySQL Router está diseñado para integrarse de forma nativa con los mecanismos de replicación y gestión de instancias de MySQL, simplificando considerablemente la configuración y la administración del balanceo de carga.

El funcionamiento de MySQL Router se basa en la detección automática de roles de las instancias (primaria y réplicas) y en la redirección transparente del tráfico de lectura y escritura. De esta manera, las operaciones que modifican datos son enviadas exclusivamente al nodo maestro, mientras que las consultas de lectura se distribuyen entre los nodos esclavos disponibles, optimizando el uso de recursos y reduciendo la carga del servidor principal. Además, MySQL Router mantiene la consistencia del estado del clúster mediante una comunicación continua con el MySQL Metadata Server, lo que garantiza la sincronización en caso de cambios de rol o fallos. [4]

Entre las principales ventajas que ofrece MySQL Router se destacan:

1. Integración nativa con MySQL, lo que elimina la necesidad de configuraciones externas o scripts complementarios.
2. Simplicidad de implementación y administración, ideal para entornos académicos, de laboratorio o de desarrollo.
3. Enrutamiento automático y transparente de consultas según el tipo de operación (lectura/escritura).
4. Alta disponibilidad y failover integrado, con conmutación automática hacia nodos replicados en caso de caída del maestro.
5. Escalabilidad horizontal, permitiendo añadir nuevas réplicas sin afectar la continuidad del servicio.
6. Compatibilidad con configuraciones locales o distribuidas y soporte directo por parte de Oracle.

Como limitación, MySQL Router está estrechamente vinculado al ecosistema MySQL, lo que restringe su uso con otros motores de base de datos.

IV. Diseño de la solución

La arquitectura propuesta se fundamenta en un esquema de replicación maestro–réplica (master–slaves) combinado con un balanceador de carga implementado mediante MySQL Router, elegida por asignación del director para el presente ejercicio práctico, con el objetivo de distribuir las operaciones de lectura y escritura de manera eficiente y garantizar la alta disponibilidad del sistema.

Arquitectura General

El sistema está conformado por cuatro componentes principales: una aplicación cliente, un MySQL Router que actúa como intermediario de conexión, un nodo maestro encargado de las operaciones de escritura y dos nodos esclavos dedicados a las lecturas. Las aplicaciones se conectan exclusivamente al Router y no directamente a los servidores MySQL; si la conexión falla, el Router reasigna el tráfico hacia otro servidor disponible, asegurando la continuidad del servicio [5].

El Router define dos rutas de conexión: una para escritura (write) y otra para lectura (read-only), separadas mediante puertos específicos (bind_port 6446 y 6447, respectivamente). Esta división garantiza un enrutamiento claro y controlado, donde las transacciones de modificación se dirigen al maestro y las consultas de lectura se reparten entre las réplicas, mejorando el rendimiento general.

La configuración del Router se adaptó de la plantilla “Basic Routing Connection” descrita en la documentación oficial de MySQL [6]. En ella, el bloque write redirige las operaciones de escritura hacia el nodo maestro utilizando la estrategia first-available, mientras que el bloque read-only distribuye las lecturas entre las réplicas mediante la estrategia round-robin.

El Router supervisa periódicamente la disponibilidad de los servidores y actualiza su tabla interna de rutas. En caso de caída de una réplica, el tráfico de lectura se redistribuye automáticamente entre las instancias restantes, lo que proporciona un equilibrio entre tolerancia a fallos en entornos locales y simplicidad al implementar la solución.

La replicación se configuró siguiendo la guía práctica descrita por Mark Drake en DigitalOcean [7], aplicando el modelo de replicación basada en binlogs, donde el maestro registra las transacciones en un log binario y las réplicas reproducen dichas operaciones para mantener la coherencia.

Cada servidor cuenta con un identificador único y rutas de log definidas para el proceso de replicación. El maestro habilita la escritura de binlogs y autoriza el acceso remoto de los esclavos, mientras que cada réplica se conecta al maestro utilizando credenciales de replicación específicas. Este esquema asegura que los datos escritos en el maestro se reflejen en las réplicas con una latencia mínima, manteniendo una consistencia eventual

adecuada para la separación de operaciones de lectura y escritura.

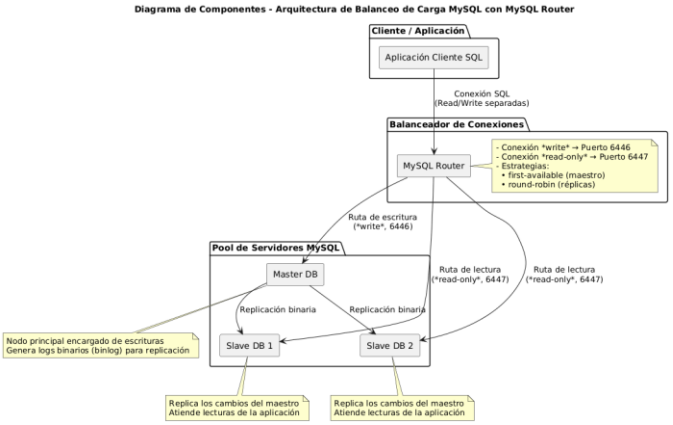
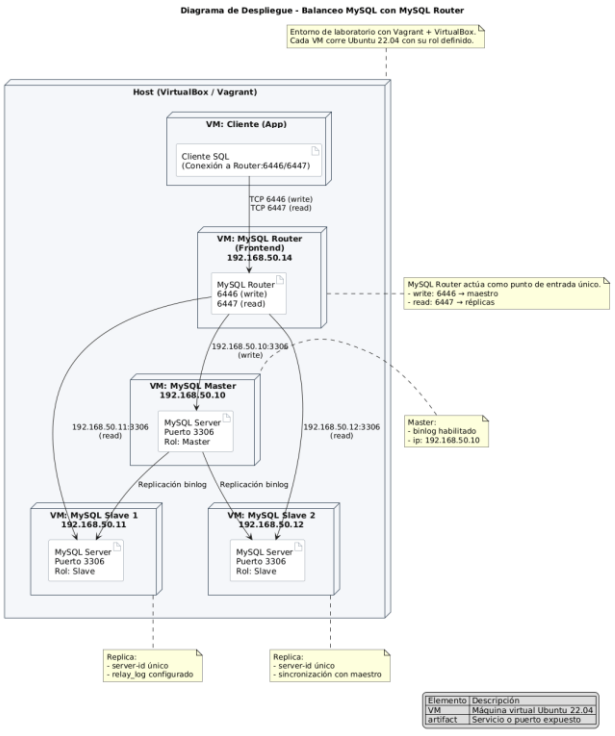


Figura 1. Diagrama de componentes de la solución.

Aspectos de Despliegue

El entorno de implementación se compuso de máquinas virtuales orquestadas con Vagrant y VirtualBox sobre Ubuntu 22.04. Las direcciones y roles fueron distribuidos de la siguiente manera:



- **Maestro:** 192.168.50.10:3306
- **Réplicas:** 192.168.50.11:3306 y, 192.168.50.12:3306
- **Router:** 0.0.0.0:6446 (escrituras) y 0.0.0.0:6447 (lecturas)

Figura 2. Diagrama de despliegue de la solución.

El Router actúa como punto de entrada único para la aplicación, redirigiendo las conexiones a los destinos configurados. Esta arquitectura modular facilita la escalabilidad mediante la incorporación de nuevas réplicas sin modificar la lógica del cliente.

V. Implementación

Entorno y topología

El desarrollo se desplegó en cuatro VMs Ubuntu 22.04 (VirtualBox + Vagrant) con una instancia source (maestro), dos replicas (esclavos) y un MySQL Router que expone dos rutas lógicas (RW/RO).

La **Tabla I** resume los roles, direcciones y puertos. Esta tabla se cita más adelante para referirse a los destinos que usa el Router.

Tabla I.Topología y roles del despliegue

Nodo	Rol	IP:Puerto	Propósito
mysql-master	Source (escrituras)	192.168.50.10:3306	Recibe todas las operaciones RW
mysql-slave1	Replica (lecturas)	192.168.50.11:3306	Backend de sólo lectura
mysql-slave2	Replica (lecturas)	192.168.50.12:3306	Backend de sólo lectura
mysql-router	Enrutador	0.0.0.0:6446 / 0.0.0.0:6447	6446=R/W → master; 6447=RO → slaves

```
STOP SLAVE;
RESET SLAVE ALL;
CHANGE MASTER TO
  MASTER_HOST='192.168.50.10',
  MASTER_USER='repl',
  MASTER_PASSWORD='replpass',
  MASTER_LOG_FILE='mysql-bin.000001',
  MASTER_LOG_POS=847;
START SLAVE;
```

Configuración de replicación

1. **Master (se debe descomentar, modificar o agregar las líneas faltantes)**
   En */etc/mysql/mysql.conf.d/mysqld.cnf* se habilitó el binlog y el acceso remoto:

```
1. server-id=1
2. log_bin=/var/log/mysql/mysql-bin.log
```

```
3. binlog_do_db=testdb
4. binlog_ignore_db=mysql
5. bind-address=0.0.0.0
```

Luego, como *root*, se creó el usuario de replicación *repl* y se obtuvieron las coordenadas del *binlog*:

- FLUSH TABLES WITH READ LOCK; (mantener la sesión abierta),
- SHOW MASTER STATUS; para capturar File y Position.

2. **Slaves**
   En cada réplica se definieron server-id={2,3}, *relay_log=/var/log/mysql/mysql-relay-bin.log* y bind-address=0.0.0.0. Tras reiniciar MySQL, se ejecutó:

La replicación usada es position-based sobre el binary log: el source registra cambios como "eventos" en el binlog y las réplicas los reproducen desde el *File/Position* capturado en el paso anterior.

TABLA II. PARÁMETROS CLAVE POR COMPONENTE

Componente	Parámetros relevantes
Master	server-id=1 log_bin=… binlog_do_db=testdb binlog_ignore_db=mysql bind-address=0.0.0.0
Slaves	server-id=2/3 relay_log=… bind-address=0.0.0.0 CHANGE MASTER TO … (File/Pos de Fig. 1)
Usuario réplica	CREATE USER 'repl'@'%' IDENTIFIED BY 'replpass'; GRANT REPLICATION SLAVE, REPLICATION CLIENT …

MySQL Router: split RW/RO y estrategias

El MySQL Router se configuró en /etc/mysqlrouter/mysqlrouter.conf con dos *routings*:

- [routing:master] bind_port=6446, mode=read-write, destinations=192.168.50.10:3306, routing_strategy=first-available.
- [routing:slaves] bind_port=6447, mode=read-only, destinations=192.168.50.11:3306,192.168.50.12:3306, routing_strategy=round-robin.

Este patrón implementa read/write splitting nativo del Router: las escrituras van al conjunto RW y las lecturas al conjunto RO.

Además, el **Connection Routing** del Router es **a nivel de conexión** (no inspecciona sentencias), lo que simplifica y maximiza el rendimiento. Inicialmente, para lecturas se usa round-robin y para escritura, first-available, estrategias soportadas oficialmente.

VI. PRUEBAS

El objetivo de esta sección es mostrar el rendimiento del sistema al realizar pruebas de carga en diferentes escenarios, como lo es el cambio de estrategia para enrutar contenido a los esclavos, o el apagado intencional de uno o más servidores.

Cabe aclarar que en las pruebas se validó tanto escritura como lectura, pero al contar con un solo maestro (único con la responsabilidad de escritura), la estrategia de enrutamiento de este servidor permanece invariante, por lo que las pruebas se centran en medir la capacidad del sistema ante eventualidades cuando se realizan operaciones de escritura.

Paso 0 – Preparación de la base de datos para pruebas

Se ejecuta el comando que permite iniciar tres tablas con 100,000 registros cada una:

```
1. sysbench /usr/share/sysbench/oltp_read_write.lua --db-driver=mysql --mysql-host=192.168.50.13 --mysql-port=6446 --mysql-user=appuser --mysql-password='1234' --mysql-db=testdb --tables=3 --table-size=100000 prepare
```

En las pruebas de carga con Sysbench para operaciones de lectura, se plantean diferentes escenarios cada uno usando cinco esclavos como réplicas (se añaden tres esclavos adicionales), usando el comando:

```
1. sysbench /usr/share/sysbench/oltp_read_only.lua --db-driver=mysql --mysql-host=192.168.50.13 --mysql-port=6447 --mysql-user=appuser --mysql-password='1234' --mysql-db=testdb --threads=64 --time=300 --report-interval=10 --percentile=99 run 2>&1 | tee sysbench_ro_64t_5m.log
```

Esto permite realizar pruebas con 64 operaciones concurrentes en el proceso de Sysbench, por un tiempo máximo de 300 segundos o 5 minutos, simulando un escenario base (escenario 0) con demanda demasiado alta, para medir el rendimiento del sistema durante uno de los peores casos.

1) Estrategia round- robin

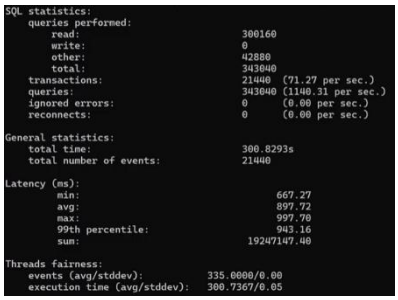


Figura 3. Estadísticas del escenario 1.

2) Estrategia first-available

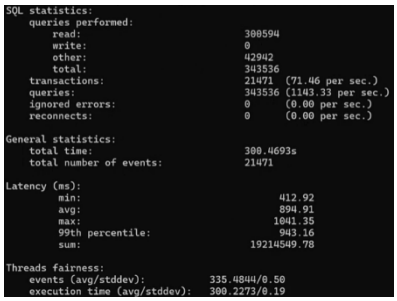


Figura 4. Estadísticas del escenario 2.

3) Master apagado y estrategia round-robin

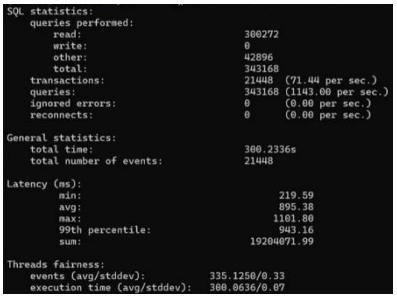


Figura 5. Estadísticas de escenario 3.

4) Slaves 2 y 5 apagados, estrategia round-robin

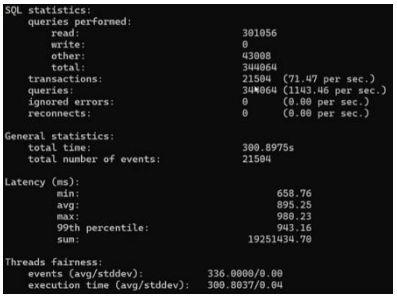


Figura 6. Estadísticas del escenario 4.

VII. Discusión de las pruebas

El escenario 0 del cual el comando de las pruebas de carga de escritura plantea, representa un caso de alta demanda que debe ser satisfecha por el sistema construido. Aunque de por sí ya configura una operación pesada, las pruebas de carga dependen en gran medida del entorno en el cual se realizan, por lo que las estadísticas pueden reflejar picos altos de demora.

No obstante, de la figura 3 podemos conocer datos interesantes: para las 21,440 transacciones SQL (cada una con 16 operaciones), cada una tomó en promedio ≈0.89 segundos en completar. Este comportamiento se explica por la alta sobrecarga sobre la máquina, hecho verificable con las métricas de "Threads Fairness", en donde se muestra que cada hilo ejecutó, del total, 335.0 transacciones exactamente (desviación estándar de 0.0). Esta uniformidad indica que todos los hilos estuvieron igualmente afectados por la demora global, y no por un hilo individual lento o esperas por bloqueos; todos los hilos permanecieron activos continuamente, aunque la ejecución real de cada uno fue intercalada por el planificador del CPU.

En el escenario que presenta la figura 4, con estrategia first-available se observaron resultados muy similares a los obtenidos con round-robin. Las latencias promedio (≈0.89 s) y el percentil 99 (≈943 ms) prácticamente no variaron, lo que sugiere un comportamiento global equivalente. Sin embargo, se aprecia una ligera dispersión en los tiempos mínimos y máximos (219 ms y 1101 ms, respectivamente), lo cual evidencia que algunos hilos accedieron a esclavos momentáneamente menos cargados, mientras que otros experimentaron esperas mayores. Las métricas de "Threads Fairness" reflejan esta pequeña variación puesto que cada hilo ejecutó en promedio 335.1 transacciones, con una desviación estándar de 0.33, frente a la uniformidad perfecta del esquema round-robin. Esta diferencia confirma que first-available distribuye las solicitudes de forma algo menos homogénea, pero sin impacto significativo en el rendimiento total, manteniéndose la demora global asociada principalmente a la contención de CPU en la máquina virtual.

La figura 5 expone el caso en el que, para operaciones de lectura, se apague el servidor master. Como se observa en las métricas, (latencia promedio: ≈895 ms), el rendimiento del sistema no se ve afectado debido a que las operaciones de lectura se relegan sobre los servidores esclavo.

Finalmente, de la figura 6 obtenemos que el rendimiento global se mantuvo prácticamente inalterado. Las latencias promedio (≈0.89 s) y percentil 99 (≈943 ms) fueron equivalentes a los casos con cinco *slaves*, mostrando estabilidad pese a la menor cantidad de nodos disponibles. Las métricas de "Threads Fairness" (336 ± 0.0 eventos) evidencian una distribución perfectamente uniforme de las transacciones, lo que confirma que el balanceador continuó asignando la carga equitativamente entre los *slaves* activos.

VIII. Conclusiones

La implementación del sistema de replicación maestro–réplica junto con el balanceo de carga mediante MySQL Router permitió distribuir correctamente las operaciones de lectura y escritura dentro del entorno planteado. La separación de rutas (lectura y escritura) funcionó de manera estable y las réplicas mantuvieron la sincronización con el maestro sin presentar inconsistencias durante las pruebas realizadas.

Las pruebas de carga con Sysbench mostraron que el rendimiento general estuvo fuertemente condicionado por las limitaciones del entorno virtualizado, especialmente por la contención de CPU. Esto produjo tiempos de respuesta similares entre todos los escenarios evaluados, con latencias promedio cercanas a 0.89 segundos y un percentil 99 estable alrededor de 943 ms. El apagado del maestro no afectó las operaciones de lectura y la desconexión de algunas réplicas no generó una

degradación apreciable del rendimiento, lo que confirma que el sistema mantuvo la disponibilidad frente a fallos simulados.

Al comparar las estrategias de enrutamiento round-robin y first-available para las operaciones de lectura, no se encontraron diferencias relevantes en las métricas obtenidas. Aunque first-available mostró una ligera variación en la distribución del trabajo entre hilos, el impacto en la latencia fue prácticamente nulo. Esto sugiere que, en este entorno específico, no hubo evidencia suficiente para afirmar que una estrategia ofrezca ventajas técnicas significativas sobre la otra (posiblemente debido a las restricciones de recursos del laboratorio utilizado para las pruebas).

En conjunto, la solución demostró ser funcional, estable y capaz de manejar escenarios de carga y fallos básicos. No obstante, para obtener conclusiones más precisas sobre el comportamiento del balanceo y las estrategias de enrutamiento, sería necesario realizar pruebas en una infraestructura con mayor capacidad y menos contención de recursos, donde los efectos del balanceo puedan observarse con claridad.

IX. Referencias

[1] Severalnines, "MySQL load balancing with HAProxy", n.d. https://severalnines.com/resources/whitepapers/mysql-load-balancing-with-haproxy/

[2] ProxySQL, "ProxySQL Cluster - ProxySQL" ProxySQL, n.d. https://proxysql.com/documentation/proxysql-cluster/

[3] ProxySQL, "How to set up ProxySQL Read/Write Split - ProxySQL". ProxySQL, n.d. https://proxysql.com/documentation/proxysql-read-write-split-howto/

[4] MySQL, "MySQL Router", n.d. https://www.mysql.com/products/enterprise/router.html

[5] MySQL, "Connection Routing", n.d. https://dev.mysql.com/doc/mysql-router/9.4/en/mysql-router-general-features-connection-routing.html

[6] MySQL, "Basic Connection Routing", n.d. https://dev.mysql.com/doc/mysql-router/9.4/en/mysql-router-deploying-basic-routing.html

[7] M. Drake, "How to set up replication in MySQL," *DigitalOcean*, Jun. 03, 2021. https://www.digitalocean.com/community/tutorials/how-to-set-up-replication-in-mysql