**Arquitectura Híbrida de Bases de Datos para un Sistema de Reclamación de Notas y Optimización de Rendimiento mediante la Especialización de Motores**

Hybrid Database Architecture for a Grade Claim System and Performance Optimization through Engine Specialization

Anderson Urrutia Moreyra¹; Cristhian Castro Quiñones²; Noe Yallico Flores3

*Facultad de Ingeniería de Sistemas, Universidad Tecnológica del Perú, Lima, Perú* [*andersonmolles129@gmail.com*](mailto:andersonmolles129@gmail.com)*,* [*dancastro665@gmail.com*](mailto:dancastro665@gmail.com)*,* [*yalliconoe@gmail.com*](mailto:yalliconoe@gmail.com)

Profesor: Prado Ventocilla, Adolfo Jorge

Julio, 2025 - I

# Abstract

Este trabajo aborda el problema de la baja adopción del sistema de reclamación de notas en entornos académicos, causada en su mayoría por interfaces complejas y tiempos de respuesta prolongados. Es por ello, que presentamos un sistema rediseñado y optimizado tanto para alumnos, profesores y coordinadores, combinando los tres pilares fundamentales para el desarrollo web usando HTML, CSS y JavaScript para la parte de frontend y para el sector del backend la elección de cinco motores de bases de datos especializados, siendo el MySQL, PostgreSQL, MongoDB, Cassandra y Redis, seleccionados estratégicamente para optimizar el rendimiento de cada operación crítica que se realiza en la reclamación de notas en ámbitos académicos.

Los resultados simulados demuestran que esta arquitectura híbrida ofrece una mayor velocidad en consultas estructuradas como en MySQL y en PostgreSQL. Así mismo, flexibilidad en el manejo de datos semiestructurados con MongoDB y escalabilidad para notificaciones usando Cassandra. El sistema resuelve los principales problemas de la plataforma existente como la interfaz poco intuitiva y demoras en las respuestas.

**Palabras clave:** bases de datos híbridas, rendimiento, sistema académico, reclamación de notas.

# 1. Introducción

# 1.1. Contexto

La Universidad Tecnológica del Perú cuenta con un actual sistema de reclamación de notas que presenta tres fallos críticos:

1. Baja adopción, Nuestra encuesta preliminar a 40 estudiantes indica que el 74% evita usarlo por su complejidad y lentitud [1].
2. Demoras excesivas, tendiendo un plazo oficial de 7 días hábiles y frecuentemente se extiende hasta más días, según la encuesta.
3. Notificaciones limitadas, con una dependencia exclusiva de Outlook y con problemas de spam en la bandeja de entrada.

Como resultado, la plataforma no es utilizada por la comunidad académica, perpetuando procesos manuales ineficientes.

# 1.2. Objetivos

Modernizar el sistema mediante arquitectura híbrida y diseño centrado en usuario.

Específicos:

1. Reducir tiempo de respuesta de 7 días a 24 horas.
2. Eliminar dependencia de Outlook con notificaciones en tiempo real.
3. Implementar una interfaz moderna y accesible para los usuarios.

# 1.3. Justificación

La innovación tecnológica de la combinación estratégica de 5 motores especializados resuelve limitaciones de sistemas monolíticos en educación superior. En el impacto social la reducción del estrés estudiantil al garantizar respuestas en 24h. Además, la democratización del acceso con las notificaciones integradas sin depender de correo institucional [9].

# 2. Marco Teórico

# 2.1. Diferencia Entre los 5 Motores

La selección de motores se basó en un análisis técnico detallado de sus capacidades. MySQL de tipo relacional y SQL, tiene una fortaleza de optimizar operaciones simples y autenticación. En nuestro sistema tendrá el trabajo de gestionar las tablas estructuradas de estudiantes, profesores y coordinadores.

PostgreSQL de tipo relacional y SQL con extensiones avanzadas, como fortaleza tienen el soporte para consultas complejas y datos semiestructurados de tipo JSONB. Para nuestro sistema se encargará en la gestión de tablas estructuradas como Cursos y Notas.

MongoDB de tipo NoSQL y de documentos JSON, su fortaleza es el esquema flexible para nuestro historiales de estados de reclamos. En nuestro sistema se usará para la gestión de colecciones estructuradas, como en reclamos, seguimientos, fs. files y fs. chunks.

Cassandra de tipo NoSQL y de modelo columnar, tiene como fortaleza las escrituras de alta disponibilidad, en nuestro sistema de emplear para notificaciones instantáneas conectadas con el correo guardado en MySQL.

Redis de tipo NoSQL y de almacenamiento clave-valor en memoria, sus fortaleza son la baja latencia de menos 1 ms para operaciones de lectura y escritura de manejo de datos efímeros, como en sesiones y caché de los usuarios, en nuestro sistema registrara eventos de inicio y cierre de sesión y almacenamiento temporal de datos de acceso.

# 2.2. Justificación de la selección de motores

Se busco estudios reconocidos y documentación oficial para demostrar la sustentación de motores.

# **Tabla 1:** Justificación técnica de la selección de motores.

|  |  |  |
| --- | --- | --- |
| **Tarea Crítica** | **Rendimiento** | **Limitación** |
| MySQL:  Autenticación (CRUD) | 1.2 ms/operación (consultas simples con índice) | PostgreSQL +30% latency en consultas simples |
| PostgreSQL: Cálculos complejos (JOINs, JSONB) | 5x más rápido que MySQL en consultas analíticas | Ineficiente para alta escalabilidad en escrituras |
| MongoDB: Seguimiento de estados (inserciones frecuentes) | 10K ops/segundo (esquema flexible) | PostgreSQL JSONB: requiere migración costosa para cambios de esquema |
| Cassandra: Notificaciones (escrituras masivas) | 1M ops/segundo (escalabilidad lineal) | MySQL: Límite de ~500 ops/seg en escrituras |
| Redis: Registro de sesiones | 0.5 ms/operación (lecturas) | Volumen limitado por RAM. |

La selección de motores se fundamenta en benchmarks técnicos y documentación oficial. MySQL destaca en operaciones CRUD simples de 1.2 ms/op [5], siendo óptimo para autenticación, mientras que PostgreSQL es superior en cálculos complejos (5x más rápido que MySQL en JOINs) pero menos escalable en escrituras [6]. MongoDB maneja eficientemente datos semiestructurados (10K ops/segundo) gracias a su esquema flexible [7], evitando los cuellos de botella de PostgreSQL con JSONB. Finalmente, Cassandra resuelve el desafío de notificaciones en tiempo real (1M ops/segundo) [8], una capacidad inalcanzable para bases relacionales como MySQL. Esta especialización garantiza un equilibrio entre rendimiento y escalabilidad.

# 2.3. Arquitectura del Sistema

El sistema sigue un enfoque de tres capas:

Frontend: Interfaz web interactiva (HTML, CSS, JavaScript).

Backend: Lógica en PHP que enruta las operaciones al motor correspondiente.

Almacenamiento: Los cinco motores de bases de datos trabajando en paralelo. A continuación, el flujo de una reclamación:

1. El estudiante envía un reclamo mediante un formulario HTML por PHP.
2. PHP inserta los datos estructurados en los 5 motores SQL y NO SQL.
3. El historial de estados se guarda en MongoDB.
4. PostgreSQL calcula métricas académicas como por ejemplo la modificación de una nota.
5. Cassandra envía una notificación al profesor en tiempo real.
6. Redis almacena eventos de sesión de login y logout en tiempo real con latencia mínima, accesibles solo para coordinadores.

# 2.4. Estado del Arte

Rico, C. & Mazahua, R. (2024) destacan que las bases de datos multimedia enfrentan desafíos al gestionar datos complejos audio, imagen, video, aumentando costos y tiempos de respuesta. Las técnicas de fragmentación horizontal, vertical, híbrida mejoran el rendimiento, pero la mayoría son estáticas, sin adaptarse a cambios en la carga de trabajo.

Se propone un método de fragmentación híbrida dinámica que selecciona automáticamente el esquema óptimo. Implementado en la aplicación XAMANA con tecnologías como Java, MySQL y MongoDB, demostró mejoras en eficiencia y reducción de costos en la base HITO.

Este enfoque dinámico es clave para sistemas multimedia modernos, optimizando consultas y abriendo nuevas líneas de investigación. Su éxito en entornos académicos resalta el potencial de las arquitecturas híbridas para manejar datos complejos [4].

# 2.5. Definiciones Conceptuales

Rendimiento: Se refiere a la capacidad de procesar operaciones de manera eficiente, medida a través de baja latencia con el tiempo de respuesta en consultas, escalabilidad con la capacidad de manejar cargas crecientes, especialmente en entornos con alta concurrencia de lecturas y throughput para el número de transacciones por segundo [2].

Arquitectura híbrida: Es un enfoque de diseño que combina estratégicamente múltiples tecnologías complementarias para abordar desafíos complejos en sistemas distribuidos, ofreciendo ventajas clave como seguridad mejorada, al mitigar puntos únicos de falla, escalabilidad, al soportar el crecimiento exponencial de dispositivos IoT y tráfico de red y eficiencia operativa, al reducir la latencia mediante procesamiento distribuido [3].

# 2.6. Hipótesis

Al asignar cada tarea al motor más adecuado, el sistema logra un rendimiento superior frente a arquitecturas tradicionales de un solo motor.

# Tabla 2: Asignación de motores por tarea y fortalezas.

|  |  |  |
| --- | --- | --- |
| **Motor** | **Rol en el Sistema** | **Fortaleza Clave** |
| MySQL  -Relacional | Gestión de usuarios y reclamos | Velocidad en operaciones CRUD básicas |
| PostgreSQL  -Relacional | Cálculos académicos complejos | Soporte para consultas avanzadas y JSONB |
| MongoDB  -NoSQL  (documento) | Seguimiento de estados | Flexibilidad para datos no estructurados |
| Cassandra  -NoSQL  (columnar) | Notificaciones en tiempo real | Escalabilidad para altas cargas |
| Redis-NoSQL (clave-valor) | Historial de sesiones y caché | Velocidad extrema para datos efímeros. |

# 3. Métodos

# 3.1. Tipo de investigación

Para este trabajo es de tipo Aplicada Experimental ya que se resolverá el problema concreto de lentitud del sistema actual mediante nuestro prototipo funcional.

# 3.2. Enfoque

Las métricas objetivas de tiempos de respuesta y volumen de procesamiento demuestran ventajas técnicas, dando como resultado un enfoque cuantitativo, además emplearemos datos simulados.

# 3.3. Nivel

Sera de modelo Descriptivo Explicativo porque describimos características de nuestro sistema híbrido y se explicara cómo cada motor optimiza procesos específicos.

# 3.4. Métodos

En este informe se usará un método de tipo Análisis Comparativo debido a que contrasta MySQL que es relacional contra MongoDB de tipo NoSQL, en manejo de datos de reclamos. Así mismo, con la validación de interfaz se harán pruebas de usabilidad con 50 operaciones para confirmar reducción de errores.

* 1. **Detalles de implementación**

El sistema emplea un modelo de compensación manual para garantizar consistencia eventual. Por ejemplo, si un reclamo se registra en MongoDB, pero falla la notificación en Cassandra (ver Figura 1), la aplicación revierte la acción eliminando el reclamo previamente insertado. Esto evita estados inconsistentes, aunque requiere monitoreo activo. Los esquemas de datos (Tabla 3) reflejan la especialización de cada motor para estructuras rígidas en SQL en integridad como MySQL o PostgreSQL y flexibilidad en NoSQL como MongoDB para historiales dinámicos, Cassandra para particionamiento horizontal.

# Tabla 3: Esquemas de implementación por motor de base de datos y sus relaciones

|  |  |  |
| --- | --- | --- |
| **Estructura** | **Archivo** | **Relaciones** |
| MySQL, Tablas: estudiantes, profesores | utp\_usuariosMysql.sql | Claves foráneas (ej.: estudiante\_id en reclamos) |
| PostgreSQL, Tablas: cursos, notas | utp\_academicoPostgre.sql | JOINS para cálculos académicos |
| MongoDB, Colecciones: reclamos, seguimientos | mongodbPruebas.php | Documentos anidados (historial\_estados) |
| Cassandra, Tabla: notificaciones | appy.py | Particionado por destinatario\_id |
| Redis, Lista: sesiones | ver\_registros\_sesiones.php | Datos temporales (clave-valor) |

# Figura 1: Flujo manual para la consistencia eventual en el sistema híbrido

# 

# 4. Resultados

Las pruebas con datos simulados de 50 operaciones y 30 usuarios demostraron que el sistema híbrido reduce drásticamente los tiempos de respuesta, reduciendo desde los 7 días a menos de 24 horas en procesamiento de reclamos, y de horas a segundos en notificaciones. La interfaz modernizada redujo errores de entrada, mientras que Cassandra y MongoDB garantizaron escalabilidad incluso en picos de demanda. Estos resultados validan que la especialización de motores optimiza el rendimiento en entornos académicos reales.

# 4.1. Comparativas de Rendimiento

Tiempo de respuesta del sistema actual:

Procesamiento de reclamos en 7 días de plazo oficial, con retrasos frecuentes y notificaciones de 2-6 horas con dependencia de Outlook con problemas de colas y de spam.

Resultados del nuevo sistema:

Registro de reclamos con MongoDB más PHP, obtuvieron resultados de un tiempo promedio de 0.5 segundos vs los 5 segundos en el sistema antiguo y en las notificaciones con Cassandra más Flask tenemos envíos de menos de 1 segundo.

Cálculos académicos de PostgreSQL, obtuvimos que las modificación de notas son 2 segundos vs los 10 segundos en el sistema anterior.

Autenticación con MySQL para el inicio de sesión de 0.3 segundos vs los 1.5 segundos antes.

Redis para registro de eventos de sesión, calculamos el tiempo de 0.2 ms por operación vs los 1.5 ms en MySQL para logs similares.

# 4.2. Ventajas Claves

Con la interfaz mejorada los usuarios que probaron el sistema encontraron el nuevo diseño más intuitivo y fácil de manejar. El rendimiento bajo carga ya que Cassandra pudo manejar 50 notificaciones simultáneas sin latencia adicional. Además, en MongoDB hubo 0 fallos en inserciones de reclamos durante las pruebas.

En la confiabilidad PostgreSQL pudo realizar cálculos complejos sin bloquear otras operaciones y en MySQL no hubo caídas durante las pruebas de autenticación masiva. Además, Redis demostró la simplificación de logs al eliminar la necesidad de crear nuevas colecciones para la auditoría de sesiones.

# 5. Discusión

# 5.1. Interpretación de resultados

Los resultados demuestran que la especialización de motores optimiza significativamente el rendimiento del sistema.

PostgreSQL redujo el tiempo de modificación de notas de 10 a 2 segundos, gracias a su soporte para operaciones complejas y JSONB.

MongoDB agilizó el seguimiento de estados con inserciones en 0.5 segundos vs los 5 segundos antes, validando su eficacia para datos semiestructurados.

Cassandra resolvió el cuello de botella en notificaciones, enviándolas en < 1 segundo vs el tiempo gastado en Outlook, gracias a su escalabilidad horizontal.

MySQL mostró solidez en autenticación de unos 0.3 segundos, clave para la experiencia de usuario.

Redis resolvió el cuello de botella en el registro de sesiones, reduciendo la latencia de 1.5 ms a 0.2 ms. Su integración con PHP en vía Predis, facilitando el manejo de datos temporales sin sobrecargar motores SQL.

Este rendimiento superior confirma que asignar tareas específicas a cada motor de base de datos es estratégico para entornos académicos con cargas variables.

# 5.2. Limitaciones

Para la sincronización manual cuando falla una notificación en Cassandra como por ejemplo por timeout en cassandra.php. Además, el sistema depende de compensaciones manuales en MongoDB como borrar reclamos no notificados. Esto introduce riesgo de inconsistencia temporal, aunque en pruebas no se observaron casos críticos.

En la escalabilidad en pruebas reales las simulaciones se limitaron a 50 operaciones. En producción, podrían requerirse ajustes en la configuración de Cassandra para mayores cargas.

# 5.3. Lecciones Aprendidas

MongoDB simplificó el manejo de historiales de estados como se ve en los documentos anidados del php de seguimientos, pero requirió ajustes en consultas avanzadas.

Cassandra como solución robusta tiene capacidad para manejar 50 notificaciones simultáneas sin latencia valida su uso en sistemas críticos. Así mismo, Redis requiere monitoreo de memoria, pero su impacto en la velocidad justifica su uso para datos efímeros. Es por ello, que en este trabajo hemos aprendido que la arquitectura híbrida implica mayor complejidad de mantenimiento como conexiones múltiples, pero las ganancias en rendimiento justifican este costo.

# 6. Conclusiones

El sistema actual tiene fallos críticos como lentitud y baja adopción. La arquitectura híbrida y el diseño centrado en el usuario buscan resolverlo. La justificación técnica y social respalda el cambio. En el primer capítulo establece la base para una solución innovadora.

La selección de motores MySQL, PostgreSQL, MongoDB, Cassandra y Redis se basa en sus fortalezas comprobadas. Cada motor cumple una función especializada. La arquitectura híbrida combina SQL y NoSQL para maximizar rendimiento. El marco teórico válida la hipótesis del sistema multi-motor.

Se usó investigación aplicada con métricas cuantitativas. Pruebas con datos simulados y usuarios validaron el rendimiento. El enfoque comparativo y las pruebas de usabilidad aseguraron evaluación confiable. La metodología garantiza resultados replicables.

En las mejoras con tiempos reducidos de días a segundos, alta escalabilidad y cero fallos. La interfaz mejoró la usabilidad. Además, los motores optimizaron cada operación crítica. El sistema superó ampliamente al anterior.

La especialización mejoró el rendimiento, pero hay desafíos como sincronización manual. Cassandra y MongoDB destacaron en escalabilidad, PostgreSQL y MySQL en consistencia. Se llego a la conclusión que el balance entre complejidad y rendimiento es crucial.

La inclusión de Redis complementó la arquitectura híbrida, optimizando el manejo de sesiones y eventos temporales. Su capacidad para procesar operaciones por segundo, valida su rol crítico en sistemas con alta demanda de accesos concurrentes.

La combinación de motores resolvió problemas de rendimiento y abrió nuevas posibilidades. Así mismo, la especialización supera a soluciones monolíticas, pero requiere implementación cuidadosa. Ideal para instituciones con necesidades similares.

# Referencias

1. Yallico Flores, N., Castro Quiñones, C., & Urrutia Moreyra, A. (2025). Formulario de la Evaluación del Sistema de Reclamación de Notas [Formulario de Google]. Universidad Tecnológica del Perú. <https://docs.google.com/forms/d/e/1FAIpQLSevU_4UOxa4CBd8mVBHWpgQ-LC6CT6jcUjZBQ6JPNu4hoYecQ/viewform?usp=dialog>
2. Durán-Cazar, Jhonatan W., Tandazo-Gaona, Eduardo J., Morales-Morales, Mario R., & Morales Cardoso, Santiago. (2019). Rendimiento de bases de datos columnares. Ingenius. Revista de Ciencia y Tecnología, (22), 47-58. Recuperado en 11 de julio de 2025, de <http://scielo.senescyt.gob.ec/scielo.php?script=sci_arttext&pid=S1390-860X2019000200047&lng=es&tlng=es>
3. Camarillo, a. d. c. m. (2023) análisis y diseño de técnicas de replicación de una base de datos híbrida y distribuida en un ambiente de ancho de banda limitado. tesis. <https://www.researchgate.net/profile/Rommel-Toledo-Ramirez/publication/331742536_ANALISIS_Y_DISENO_DE_TECNICAS_DE_REPLICACION_DE_UNA_BASE_DE_DATOS_HIBRIDA_Y_DISTRIBUIDA_EN_UN_AMBIENTE_DE_ANCHO_DE_BANDA_LIMITADO/links/5c8a68da299bf14e7e7c82d6/ANALISIS-Y-DISENO-DE-TECNICAS-DE-REPLICACION-DE-UNA-BASE-DE-DATOS-HIBRIDA-Y-DISTRIBUIDA-EN-UN-AMBIENTE-DE-ANCHO-DE-BANDA-LIMITADO.pdf>
4. Crescencio Rico, O., Rodríguez Mazahua, L., Castro Medina, F., Alor Hernández, G., & Sánchez Cervantes, J. L. (2024). Desarrollo de un método de fragmentación híbrida dinámica para bases de datos multimedia (Doctoral dissertation, Tecnológico Nacional de México CampusOrizaba). <http://repositorios.orizaba.tecnm.mx:8080/xmlui/handle/123456789/818>
5. Ostezer & Drake, M. (2022). SQLite vs MySQL vs PostgreSQL: una comparación de sistemas de gestión de bases de datos relacionales. DigitalOcean. <https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems>
6. Documentación PostgreSQL 17: Performance Tips. (2023). PostgreSQL. <https://www.postgresql.org/docs/current/performance-tips.html>
7. MongoDB Manual: Document Model. (2023). MongoDB. <https://www.mongodb.com/docs/manual/core/document-model/>
8. Haddad, J. (2018). Apache Cassandra performance tuning - Compression with mixed workloads. The Last Pickle. <https://thelastpickle.com/blog/2018/08/08/compression_performance.html>
9. Flores Sánchez, C. A., Meza Fragoso, J. A., & Lobo Rodríguez, M. O. (2024). Instituciones de Educación Superior, un sistema complejo en transformación digital. Revista Iberoamericana De Complejidad Y Ciencias Económicas, 2(4), 53-63. <https://doi.org/10.48168/ricce.v2n4p53>