**1. Resumen Ejecutivo**

* Crecimiento exponencial de la plataforma de streaming ha llevado a DB relacional monolítica a su límite. Latencias inaceptables en consultas de historial y recomendaciones, lo que degrada la experiencia del usuario y pone en riesgo capacidad para competir.
* Se propone una modernización de la infraestructura de datos. Se recomienda usar la mejor base de datos NoSQL para cada tarea específica: una **base de datos de documentos (MongoDB)** para perfiles y catálogo, una **base de datos columnar (Cassandra)** para el historial de visualización, y una **base de datos de grafos** para el motor de recomendaciones en tiempo real.
* Esta estrategia permitirá escalar cada componente del sistema de forma independiente, lograr un rendimiento óptimo en todas las áreas de la aplicación y reducir costos operativos.

**2. Análisis del Problema**

* **Ineficiencia para Datos de Series Temporales:** El **historial de reproducción** es una serie temporal que crece indefinidamente. En una DB SQL, consultar el historial de un usuario con millones de registros requiere escanear índices masivos, lo cual es lento e ineficiente.
* **Incapacidad para Consultas de Grafos:** Las **recomendaciones personalizadas** (ej. "usuarios que vieron X también vieron Y", "amigos de tus amigos que vieron Z") son intrínsecamente un problema de grafos. Intentar resolver esto con JOINs recursivos en SQL es extremadamente costoso en términos de cómputo y no funciona en tiempo real.
* **Cuello de Botella Centralizado:** Aunque los perfiles de usuario y el catálogo de contenido podrían funcionar en una DB relacional, están contenidos dentro de la misma base de datos sobrecargada por el historial y las recomendaciones. Un peak de tráfico en el historial de visualización puede hacer que la plataforma, se vuelva lenta o se caiga directamente.
* **Escalabilidad Costosa y Compleja:** Como se menciona en el caso, escalar horizontalmente es costoso para este tipo de base de datos.

**3. Comparación de Tecnologías**

* **Análisis Comparativo por Caso de Uso:**

|  |  |  |  |
| --- | --- | --- | --- |
| Tipo de Base de Datos | Mejor Aplicación en el Caso de Uso | ¿Por Qué? | Ejemplos |
| **Document-Oriented** | **Perfiles de Usuario y Catálogo de Contenido** | Su esquema json es perfecto para almacenar perfiles con atributos variados y metadatos complejos de películas o series. | MongoDB |
| **Column-Oriented** | **Historial de Reproducción y Eventos** | Está optimizada para manejar cantidades masivas de datos de series temporales. | Cassandra |
| **Graph-Oriented** | **Motor de Recomendaciones Personalizadas** | Su propósito es modelar y consultar relaciones complejas. Preguntas como "¿qué contenido ven los amigos de este usuario?" se resuelven de forma nativa y en tiempo real. | Neo4j |
| **Key-Value Store** | **Caché de Sesiones y Datos Populares** | Ofrece la latencia más baja posible para consultas simples (clave-valor). Es ideal como una capa de caché para almacenar tokens de sesión de usuario o los metadatos del contenido más popular por ejemplo | DynamoDB |

**4. Propuesta de Solución**

* **Justificación:**
  + El error fundamental del sistema actual es tratar de resolver problemas fundamentalmente diferentes con una sola herramienta. Una arquitectura moderna y escalable reconoce que cada tipo de datos y patrón de acceso requiere una solución optimizada.
* **Diseño de la Arquitectura Propuesta:**
  1. **Servicio de Cuentas de Usuario:** Utilizará **MongoDB** para gestionar toda la información de perfiles, preferencias, listas de favoritos y configuraciones.
  2. **Servicio de Historial:** Utilizará **Cassandra** para registrar cada evento de reproducción.
  3. **Motor de Recomendaciones:** Utilizará **Neo4j**. Los datos de usuarios, contenido y visualizaciones se cargarán en el grafo para modelar las relaciones.
  4. **Capa de Caché:** Se implementará **DynamoDB** para almacenar datos de acceso frecuente, como las sesiones de los usuarios activos, para una autenticación y carga de perfil casi instantánea.

**5. Conclusiones y Recomendaciones Finales**

* **Conclusión:** La adopción de una arquitectura de diferentes tipos es una evolución necesaria para la plataforma. No solo resuelve problemas actuales de rendimiento y escalabilidad, sino que proporciona una base flexible y robusta para la innovación futura.
* **Recomendaciones:**
  1. **Reestructuración de Equipos:** Organizar los equipos de desarrollo en torno a los microservicios propuestos.
  2. **Implementación por Fases:** Iniciar el proyecto migrando el **Servicio de Historial a Cassandra**, ya que es la mayor fuente de datos y probablemente el principal cuello de botella.
  3. **Invertir en Capacitación:** Proveer al equipo los recursos necesarios para formarse en las nuevas tecnologías y en los principios de diseño de sistemas distribuidos.