Caso de uso de Modelos NoSQL para una red social

Sebastián Jaimes, Leonard Kennedy
Juan Borquez, Juan Camilo Bohórquez Pérez

La Salle Barcelona - Universitat Ramon Llull, 2024

ÍNDICE

1. CASO DE USO DE LA RED SOCIAL Y ARGUMENTACIÓN DE LA DB NOSQL

1.1. DESCRIPCIÓN DEL CASO DE USO

La red social que estamos considerando permite a los usuarios publicar mensajes, compartir fotos, seguir a otros usuarios, dar 'me gusta' a publicaciones y comentar en ellas. Además, los usuarios pueden tener perfiles personalizados con información como nombre, edad, ubicación, intereses, etc. Esta red social está experimentando un rápido crecimiento de usuarios y contenido.

1.2. ARGUMENTACIÓN DE LA DB NOSQL

Una base de datos NoSQL es la opción ideal para esta red social por varias razones:

- **Escalabilidad horizontal:** A medida que la red social crece y el número de usuarios y datos aumenta exponencialmente, una base de datos NoSQL puede escalar horizontalmente sin problemas, distribuyendo la carga de trabajo en varios servidores.
- **Modelo de datos flexible:** La estructura de datos en una red social puede ser bastante variable y no necesariamente tabular. Con una base de datos NoSQL, podemos almacenar datos en forma de documentos (en el caso de una base de datos orientada a documentos como MongoDB), lo que permite manejar datos heterogéneos y cambiar el esquema de manera ágil.
- **Rendimiento:** Las consultas en una red social suelen ser complejas y deben ejecutarse rápidamente para proporcionar una experiencia de usuario fluida. Las bases de datos NoSQL están optimizadas para consultas rápidas en grandes volúmenes de datos, lo que garantiza un rendimiento óptimo incluso en momentos de alta carga.

2. TEOREMA CAP

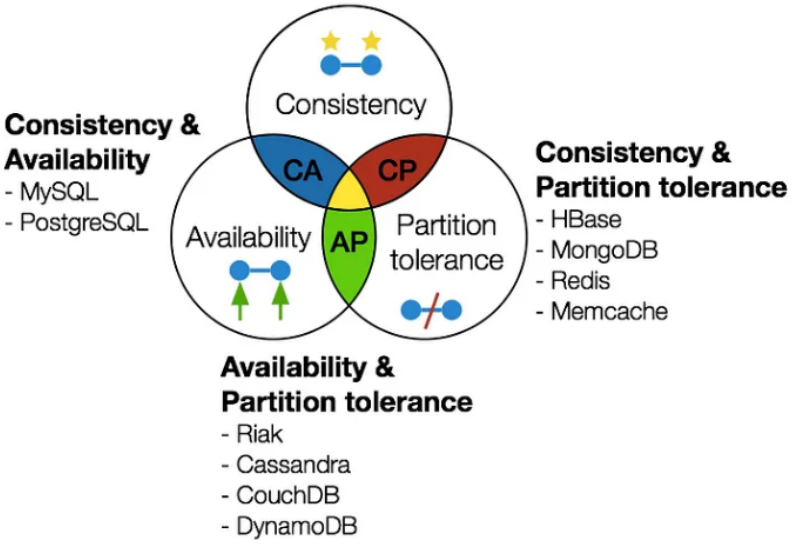
Para comprender mejor el Teorema CAP, es esencial examinar detalladamente cada una de sus características fundamentales. Estas características son:

- **Consistencia:** Implica que cualquier modificación realizada en los datos del sistema debe ser aplicada de manera consistente en todos los nodos, asegurando que la información sea la misma en cada uno de ellos. La consistencia busca evitar discrepancias o divergencias en los datos entre los distintos nodos del sistema distribuido.
- **Disponibilidad:** Garantiza que cada solicitud enviada a un nodo reciba una confirmación de si se ha ejecutado satisfactoriamente o no. La disponibilidad se centra en mantener activo y

respondiendo al sistema, asegurando que los usuarios o servicios puedan acceder y realizar operaciones, incluso en situaciones de fallos o particiones.

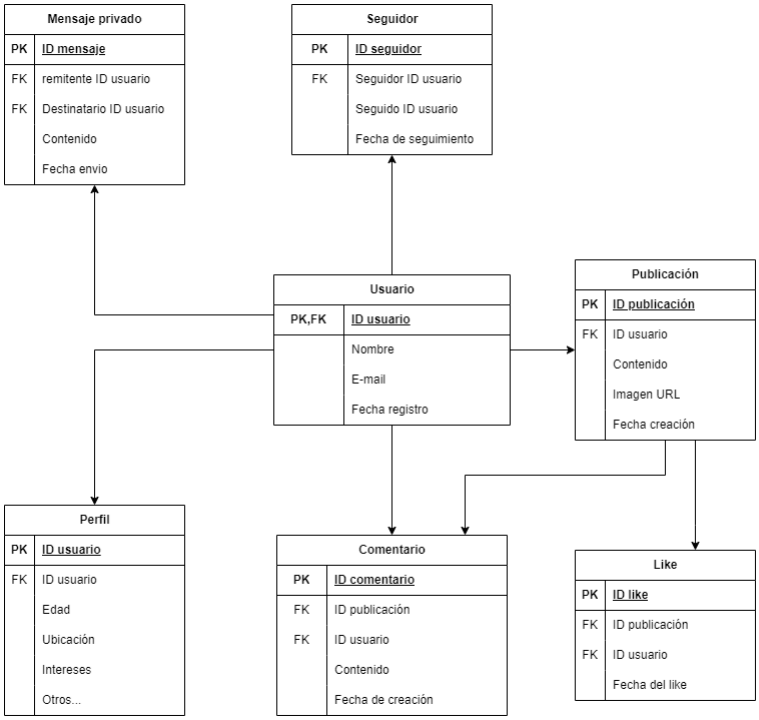
- **Tolerancia al Particionamiento:** Implica la capacidad del sistema para continuar funcionando incluso si se producen fallos de comunicación entre nodos, como particiones de red. La tolerancia al particionamiento asegura que el sistema permanezca operativo a pesar de la separación temporal o permanente de nodos, independientemente de la causa de la desconexión.

Ahora que hemos comprendido las características del Teorema CAP (Consistency, Availability, Partition Tolerance), recordemos que este teorema establece la imposibilidad de que un sistema distribuido ofrezca simultáneamente las tres características mencionadas.
En el contexto de la base de datos NoSQL MongoDB, que será la encargada de gestionar la red social en nuestro caso de uso, examinaremos la situación con la siguiente imagen:



En este esquema, MongoDB Utiliza un modelo (CP), permitiendo que los nodos de la red social puedan operar de manera independiente incluso en situaciones de partición de red. La arquitectura de réplicas en MongoDB asegura la disponibilidad, ya que los datos se replican en varios nodos, permitiendo que el sistema siga respondiendo incluso si algunos nodos experimentan fallas o pérdida de conexión.

3. MODELO LÓGICO DE DATOS DE LA RED SOCIAL



4. FACTORES QUE AFECTAN LA BASE DE DATOS

En una base de datos no relacional de una red social construida con MongoDB, existen algunos factores que podrían provocar lentitud:

1. **Índices inadecuados:** Puede haber un impacto negativo en el rendimiento de la base de datos si no se utilizan los índices adecuados en las consultas, por ejemplo en campos utilizados para filtrar o buscar datos.

   - **Caso:** Si no se crea un índice en el campo de fecha de publicación, las consultas que buscan publicaciones recientes pueden volverse lentas a medida que aumenta el volumen de datos. Lo mismo podría suceder con el índice adecuado para el nombre de usuario.
   - **Solución:** En MongoDB, es posible utilizar el método 'createIndex' para crear índices en los campos más relevantes para las consultas más frecuentes. Cabe destacar que la creación de índices también tiene implicaciones en el almacenamiento y el rendimiento de escritura. Por lo que la selección de campos relevantes es de suma importancia. También, es posible identificar cuando las consultas son lentas mediante el uso del

'Profiler', que permite identificar qué consultas están afectando el rendimiento de la base de datos y hacer ajustes en los índices según sea necesario.

2. **Consultas complejas de búsqueda:** El rendimiento de la base de datos se puede ver afectado si se realizan consultas complejas para buscar publicaciones, comentarios o perfiles de usuarios, que involucran múltiples operaciones y filtros, puede generar una carga significativa en la base de datos y ralentizar el rendimiento.

   - **Caso:** Una consulta que busca todas las publicaciones de un usuario específico, que tenga muchos comentarios y reacciones podría requerir mucho tiempo de procesamiento y afectar el rendimiento general.
   - **Solución:** Es posible utilizar la herramienta 'explain()' para analizar el plan de ejecución de la consulta. Esta herramienta proporciona información detallada sobre cómo se ejecuta la consulta, identifica si utiliza índices y cómo los utiliza. Esto permitirá identificar posibles cuellos de botella y optimizar las consultas que pueden llegar a ser problemáticas.

3. **Escalabilidad insuficiente:** Otro factor que puede provocar lentitud es la escalabilidad, si el clúster de MongoDB no tiene suficientes réplicas o no se ha implementado el particionamiento adecuado.

   - **Caso:** La red social se vuelve cada vez más popular, crece y el número de usuarios y las publicaciones aumentan.
   - **Solución:** Escalar horizontalmente el clúster de MongoDB, agregando más réplicas y distribuyendo la carga de trabajo de manera equitativa. Además, se puede implementar el particionamiento de datos para distribuir los datos en diferentes servidores y evitar cuellos de botella.

4. **Falta de caché:** Sin caché, cada vez que una aplicación realiza una consulta, debe acceder a la fuente de datos original, lo que puede resultar en tiempos de respuesta más lentos y un mayor consumo de recursos. Esto puede afectar negativamente la experiencia del usuario y la eficiencia de la base de datos.

   - **Caso:** Cuando un usuario inicia sesión en la red social y visita su perfil, se realiza una consulta a la base de datos para obtener la información del perfil y luego otro usuario visita ese mismo perfil.
   - **Solución:** Implementar una capa de caché utilizando las herramientas 'Memcached' o 'Redis' para mejorar el rendimiento y la velocidad de respuesta de la base de datos. Se almacenan en caché los resultados de las consultas frecuentes de los perfiles de los usuarios, se evita la necesidad de realizar consultas repetitivas a la base de datos, reduciendo la carga en el sistema y mejorando la eficiencia. También, se puede utilizar TTL (Time-to-Live), que permite controlar la vigencia del caché, asegurándose de que los datos se estén actualizando correctamente y evitando la obsolescencia de la caché.