Bases de datos NoSQL
Clase 1

Arroyo Joaquin
Belmonte Marina

Universidad Nacional de Rosario
Licenciatura en Ciencias de la Computación
Bases de Datos Avanzadas

17 de abril de 2024

Resúmen

- Caracterización de Bases de Datos NoSQL
- Razones de su surgimiento
- SQL versus NoSQL
- Sistemas Distribuidos
- Clasificación de modelos NoSQL

Caracterización

¿ Qué es una base de datos *NoSQL* ?

La definición más simple es: "*Una base de datos que no sigue el modelo relacional*" .

Una gran cantidad de modelos se adaptan a dicha definición, es por esto que además presentan las siguientes características:

- Pueden correr fácilmente en clusters sin *Single Point of Failure*.
- Operan sin un esquema fijo, permitiendo agregar campos a los registros *on-the-fly*.
- Siguen el principio **BASE** (**B**asically **A**vailable, **S**oft State, **E**ventually Consistent)
- Hacen incapié en el desempeño y la flexibilidad sobre la potencia de modelización y las consultas complejas.
- Usualmente son nuevas y de software libre.

¿Por qué NoSQL?

Pero, ¿por qué surgen las bases de datos no relacionales?

Principalmente por el crecimiento exponencial de la información, debido al uso de nuevas tecnologías en la sociedad.

Dicho crecimiento es impulsado por

- Internet
- Teléfonos móviles
- Redes Sociales
- etc.

Como consecuencia surgieron conceptos como

- Web 2.0
- Big Data
- Cloud Computing

¿Por qué NoSQL?

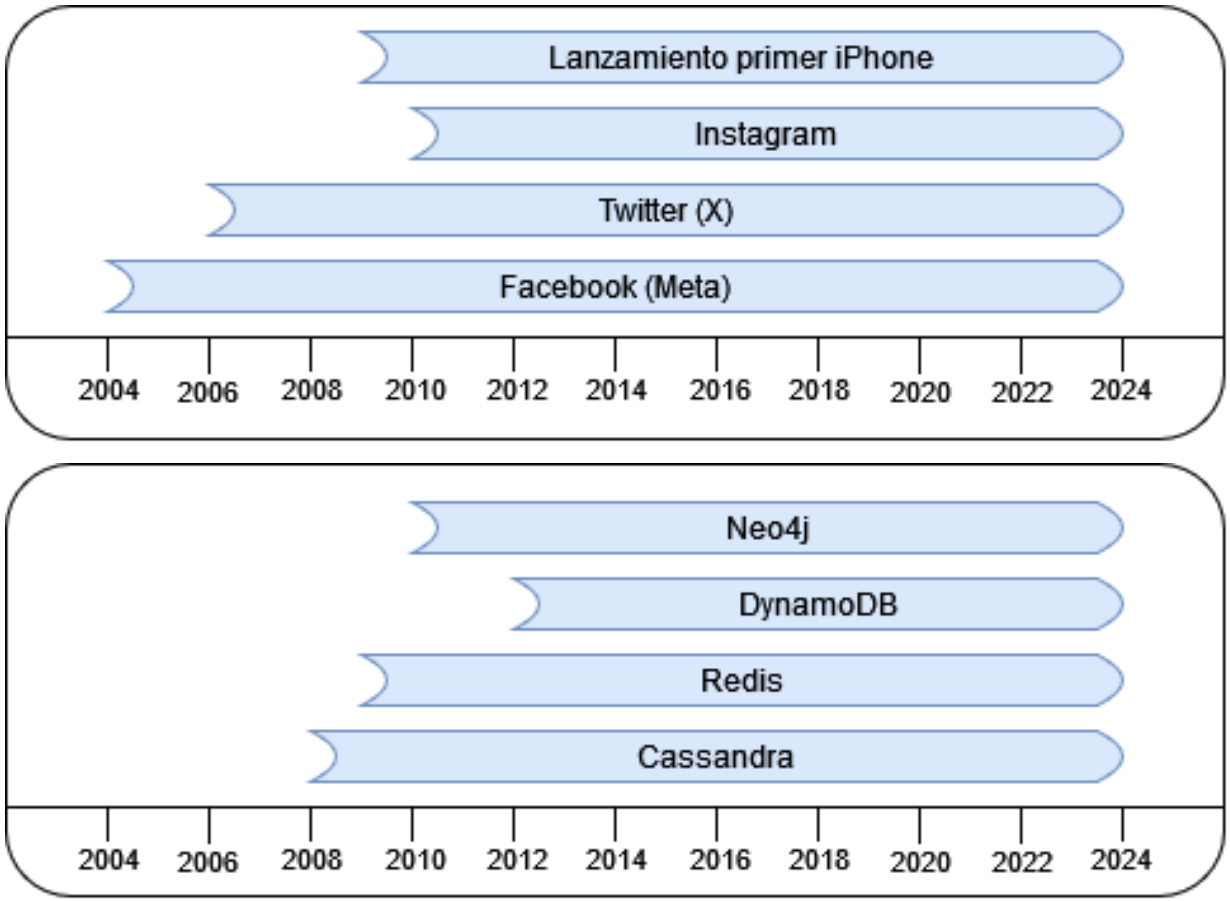
Estos avances llevaron tanto a la industria como a la academia a replantearse el uso de los modelos relacionales debido a sus limitaciones naturales en el manejo de grandes cantidades de información.

Se necesitaban (*necesitan*) sistemas que escalen horizontalmente de manera sencilla.

Se empiezan a investigar y desarrollar posibles alternativas por fuera de los modelos relacionales, que resuelvan estos problemas.

Prinpalmente es por esto que surgen las *Bases de Datos NoSQL*.

¿Por qué NoSQL? - Tecnologías vs NoSQL



SQL vs NoSQL

Las principales diferencias entre estos dos modelos de datos son

1. **Estructura de los datos.**
   SQL usa el modelo relacional, esquemas rígidos.
   NoSQL no usa esquemas rígidos, da más flexibilidad.
2. **Escalabilidad**
   SQL escala mejor verticalmente.
   NoSQL escala mejor horizontalmente.
3. **Consistencia**
   SQL garantiza consistencia en todas sus transacciones.
   NoSQL prioriza tolerancia a particiones y disponibilidad sobre consistencia.

Sistemas Distribuidos - Modelos de Replicación

Una de las características de los sistemas NoSQL es la facilidad de correr en sistemas distribuidos, lo que permite el escalado horizontal.

Generalmente se realiza cuando el sistema está operativo.

Se necesitan técnicas para distribuir la información actual entre los nuevos nodos sin interrumpir al sistema.

Para resolver este problema están los

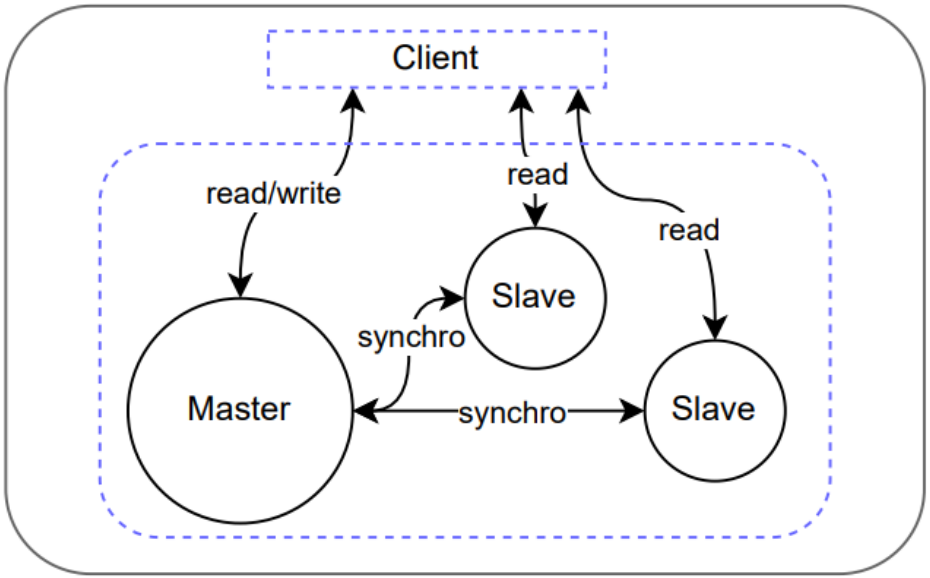
Modelos de Replicación

Los dos principales son **Maestro-Esclavo** y **Maestro-Maestro**

Sistemas Distribuidos - Modelo Maestro-Esclavo

En este modelo tenemos un **Maestro** y uno o más **Esclavos**.

- **Maestro.**
  1. Es el nodo principal del sistema.
  2. Responsable de aceptar escrituras en la base de datos.
- **Esclavo.**
  1. Es un nodo secundario del sistema.
  2. Se sincroniza con el maestro para mantener una copia de los datos.
  3. Solo acepta lecturas.

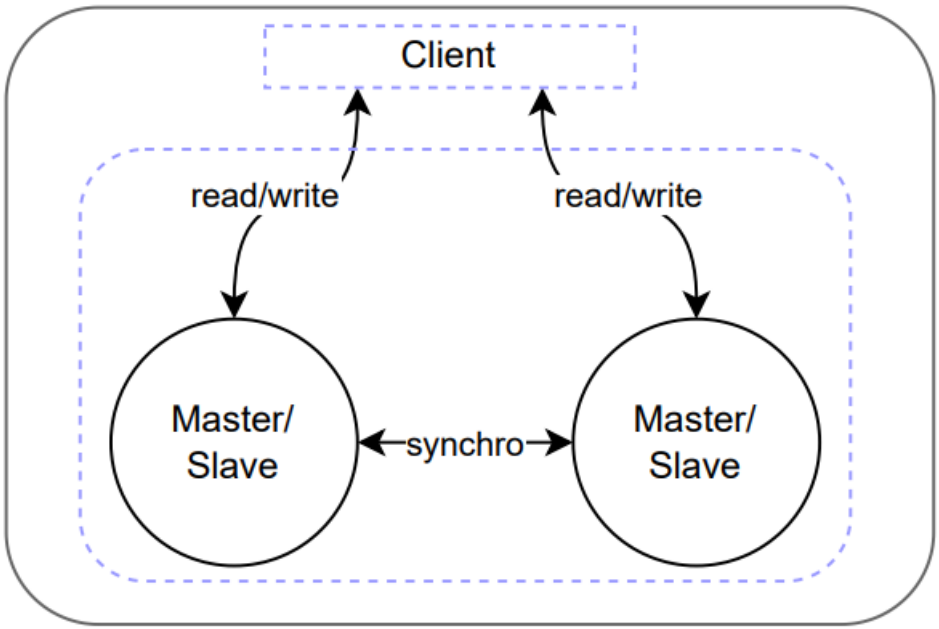


Sistemas Distribuidos - Modelo Maestro-Maestro

En este modelo todos los nodos en el sistema son tanto **Maestros** como **Esclavos**.

La replicación de datos es bidireccional.

Cada nodo es responsable de mantener una copia exacta de los datos y asegurar la sincronización.



Sistemas Distribuidos - Modelos de Replicación

La principal diferencia entre los modelos, es que el modelo **Maestro-Esclavo** cuenta con un único nodo **Maestro**.

Por lo que podemos decir que dicho modelo cuenta con un control *Centralizado*.

Este nodo se puede convertir en un cuello de botella. Y además, si falla el nodo Maestro... estamos en problemas.

Igualmente, existen soluciones para este problema

1. Tener uno o más nodos Maestros de backup.
2. Asignar hardware potente al nodo Maestro.
3. etc.

Estas soluciones pueden combinarse.

Sistemas Distribuidos - Partición de Archivos

La Partición de Archivos es otra característica a tener en cuenta en los sistemas distribuidos.

Se refiere a la división de la información en fragmentos que se distribuyen entre los nodos.

Esta distribución mejora el rendimiento al permitir a su vez distribuir las consultas y hacer paralelismo.

Aumenta la resilencia del sistema debido a la redundancia de la información, si falla algún nodo, la información sigue disponible.

Sistemas Distribuidos - Acceso de Datos de Alto Desempeño

Generalmente se necesita encontrar un registro particular entre millones.

El crecimiento de la información hace que esta búsqueda se ralentice.

Aquí entra en juego el *High Performance Data Access* (HPDA).

Se refiere a la capacidad de un sistema para acceder y manipular grandes volúmenes de datos de manera rápida y eficiente.

Se suelen utilizar técnicas de *hashing* y/o particionado de rango sobre claves de valores.

Sistemas Distribuidos - Acceso de Datos de Alto Desempeño

- **Hashing.**
  Esta técnica aplica una función de hash h a una clave K.
  La aplicación $h(K)$ devuelve la ubicación del objeto en el sistema distribuido.
- **Particionado de rango sobre claves.**
  Se asignan los valores a ubicaciones en función de un rango de valores de clave.
  Una ubicación u podría tener los valores cuyas claves K estén en el rango

$$Ku_{min} \leq K \leq Ku_{max}$$

Sistemas Distribuidos - Conjetura de Brewer

También conocida como Teorema CAP.

Las siglas **CAP** vienen de:

- **C***onsistency* (Consistencia)
- **A***vailability* (Disponibilidad)
- **P***artition Tolerance* (Tolerancia a Particiones)

Dicho Teorema enuncia lo siguiente

Teorema

Dadas las propiedades **Consistencia**, **Disponibilidad** *y* **Tolerancia a Particiones**, *solo se pueden cumplir dos de estas tres simultáneamente en un sistema distribuido.*

Los sistemas NoSQL, sacrifican la consistencia por *consistencia eventual*, para garantizar en mayor medida disponibilidad y tolerancia a particiones.

Clasificación

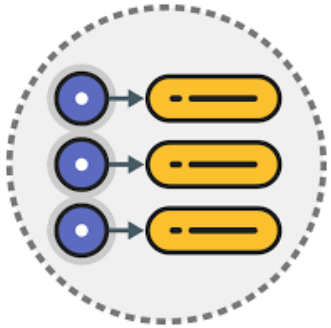
Existe una amplia variedad de modelos de datos *NoSQL*. Entre los más conocidos encontramos:

1. Clave-Valor
2. Columnar
3. Documental
4. Grafo

Índice

Clave-Valor

Este modelo almacena pares del tipo (*Key*, *Value*).



- **Key**: Un identificador único para acceder al valor asociado.
- **Value**: Puede ser cualquier tipo de dato, desde texto, números y documentos, hasta listas o incluso otros pares clave-valor.

Clave-Valor - Ventajas

- No hay una estructura de tabla rígida, que permite esquemas flexibles.
- Los campos pueden añadirse y modificarse dinámicamente sin alterar el esquema de la base de datos.
- Permiten el funcionamiento en clústeres, facilitando la adición de nodos para manejar grandes volúmenes de datos.
- Los motores de procesamiento en paralelo y la arquitectura de clústeres reducen los tiempos de consulta y mejoran la velocidad de escritura y lectura.

Clave-Valor - Ejemplos

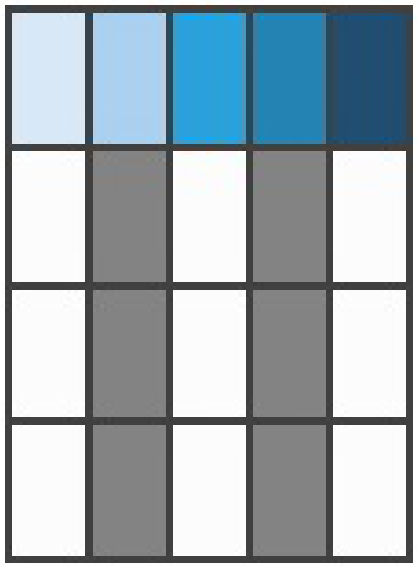
Algunas de las implementaciones más conocidas son:

- **Amazon DynamoDB**
  Una base de datos clave-valor y documental de Amazon Web Services. Ofrece alta escalabilidad y disponibilidad.
- **Voldemort**
  Una base de datos distribuida impulsada por LinkedIn, diseñada para ofrecer alta disponibilidad y escalabilidad.
- **Redis**
  Un proyecto de código abierto muy utilizado, especialmente para aplicaciones web modernas. Soporta estructuras de datos avanzadas como listas, sets y hashes.
- **Riak**
  Un sistema escalable que facilita el desarrollo ágil de aplicaciones, con un enfoque en la simplicidad de uso.

Índice

Columnar

Las bases de datos orientadas a columnas almacenan datos verticalmente por columnas en lugar de horizontalmente por filas, permitiendo un acceso más eficiente a datos específicos.



Columnar - Ventajas

- Mayor eficiencia en consultas sobre columnas específicas.
- Convenientes para análisis de grandes volúmenes de datos.
- Adecuadas para aplicaciones de business intelligence y análisis predictivo.

Columnar - Desventajas

- Operaciones de escritura más lentas debido a la reorganización de datos en columnas específicas.
- Posibles desafíos en entornos con datos altamente transaccionales.

Columnar - Ejemplos

Algunos ejemplos de bases de datos columnares son:

- **Apache Cassandra**
  Aunque es principalmente una base de datos de clave-valor, Cassandra utiliza un modelo de almacenamiento columnar para mejorar el rendimiento de ciertas consultas.
- **Apache HBase**
  HBase es una base de datos de columnas distribuida, que almacena datos de forma columnar y permite un acceso eficiente a través de claves de fila.
- **ClickHouse**
  ClickHouse es una base de datos de análisis columnar de código abierto diseñada para consultas analíticas de alto rendimiento en grandes conjuntos de datos.

Índice

Documental

Las bases de datos documentales almacenan datos en documentos individuales, que pueden ser estructurados o semi-estructurados como archivos JSON o XML.

Documental - Ventajas

- Las bases de datos son flexibles.
- Los sistemas pueden crear índices sobre algunos elementos de datos para mejorar el rendimiento de las consultas.

Documental - Casos de uso

Las bases de datos documentales son especialmente adecuadas para aplicaciones donde los datos tienen una estructura flexible y variable como por ejemplo:

- Contenido web
- Análisis de registros
- Gestión de datos de productos

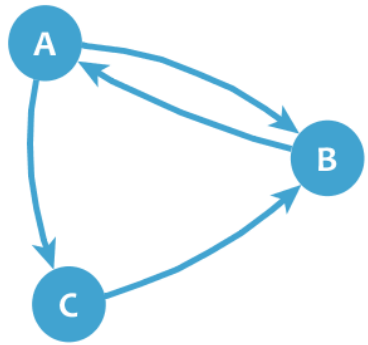
Documental - Ejemplos

- **MongoDB**
  Es una base de datos documental líder que utiliza un modelo de datos basado en documentos JSON. Ofrece capacidad de escalabilidad horizontal y opciones avanzadas de replicación, lo que permite gestionar grandes cantidades de datos y cargas de trabajo exigentes.
- **CouchDB**
  Se destaca por su simplicidad en la replicación y consulta. Utiliza un modelo de datos basado en documentos JSON y es especialmente adecuado para cargas de trabajo menos intensivas y entornos donde la replicación descentralizada es una necesidad.

Índice

Grafo

Las bases de datos en grafo utilizan estructuras de grafo para almacenar, consultar y relacionar datos.



Los datos se representan mediante nodos (entidades) y arcos (relaciones entre entidades).

Pueden asignarse propiedades a nodos y arcos para capturar más detalles.

Grafo - Casos de uso

Las bases de datos en grafo son ideales para almacenar datos interconectados. Aplicaciones comunes incluyen:

- Redes sociales
- Sistemas de recomendación
- Redes de trasportes

Los datos almacenados pueden interpretarse de diferentes maneras basadas en las relaciones entre los nodos.

Grafo - Desventajas

- Las bases de datos en grafo presentan complejidad de modelado y un costo de aprendizaje elevado para usuarios no familiarizados.
- Pueden experimentar dificultades de rendimiento y requerir esfuerzos adicionales de mantenimiento y optimización.

Grafo - Ejemplos

Algunos ejemplos de bases de datos en grafos son:

- **Neo4J**
  Una de las bases de datos en grafo más populares y utilizadas en la industria. Ofrece una gran variedad de herramientas y bibliotecas para el modelado y análisis de grafos.
- **GraphBase**
  Un sistema que permite crear y consultar grafos de datos, con capacidades avanzadas para el análisis de datos.
- **Infinite Graph**
  Ofrece una plataforma para trabajar con grafos a gran escala, con soporte para consultas complejas y análisis de grafos.
- **FlockDB**
  Un sistema de bases de datos en grafo diseñado para manejar relaciones entre grandes volúmenes de datos, con un enfoque en el rendimiento y la escalabilidad.

¿Dudas?

Referencias

- N. Reyes, O. E. Gagliardi, C. Deco, and M. T. Taranilla. *Bases de Datos, Maestría en Tecnología de la Información*. 2018.
- C. M. Bender, C. Deco, J. S. González, M. Hallo, and J. C. Ponce Gallegos. *Tópicos Avanzados de Bases de Datos*. 2014.
- E. Gómez and L. Mota. *Introducción a las bases de datos NoSQL. Sistemas de bases de datos orientados a grafos*. 2021.