TRABAJO DE INVESTIGACIÓN

APACHE CASSANDRA

[Base de datos NoSQL]

LINK VIDEO:
https://youtu.be/TnwqlQcC42g

ALUMNOS

Campott, Daiana
Carlini, Gianluca
Gibon, Florencia
Nesteruk Bonazzola, Aldana
Paladini, Nicolás

CARRERA: Analista Programador

COMISIÓN: 2K

AÑO: 2023

DOCENTE: María Roxana Martínez

MATERIA: Base de datos

UAI
Universidad Abierta Interamericana

Índice

Introducción……3
¿Qué son las bases de datos NoSQL?.....5
Diferencias entre BBDD SQL y NoSQL……6
Tipos de BBDD NoSQL……7
Motores de BBDD NoSQL……...9
Cassandra……12
Ventajas y Desventajas de los sistemas NoSQL, con énfasis en Apache Cassandra……14
Teorema CAP……..17
Características de Apache Cassandra……19
Tipos de consulta CQL……...22
Ejemplos de consultas CQL……..23
Cassandra y sus Componentes……24
Cassandra Proceso de Escritura y Lectura……26
Conclusión……29
Opiniones de porque usariamos Cassandra……..29
Bibliografía……31

Introducción

Las bases de datos han sido fundamentales en el desarrollo del mundo de la informática, permitiendo el almacenamiento y recuperación eficiente de grandes volúmenes de datos. Sin embargo, con el rápido crecimiento de las aplicaciones web y móviles, surgió la necesidad de manejar datos no estructurados o semi estructurados de manera eficiente. Aquí es donde entran en juego las bases de datos NoSQL.

Las bases de datos Not Only SQL son sistemas de gestión de bases de datos que ofrecen una alternativa al modelo tradicional de bases de datos relacionales. A diferencia de las bases de datos SQL, que siguen una estructura rígida y predefinida, las bases de datos NoSQL son flexibles y pueden adaptarse fácilmente a diferentes tipos de datos.

Uno de los motores de bases de datos NoSQL más populares es Apache Cassandra. Cassandra es un sistema altamente escalable y tolerante a fallos, está diseñado para manejar grandes volúmenes de datos sin comprometer la disponibilidad.

Fue desarrollada por Avinash Lakshman y Prashant Malik en Facebook y publicada por primera vez en 2008. En el año 2009, la Apache Software Foundation, una de las comunidades de desarrolladores de código abierto más importantes, adquirió el proyecto como subproyecto de Apache Incubator. Desde febrero de 2011, Apache Cassandra ya es un proyecto independiente de máximo nivel dentro de la Apache Software Foundation, como el extendido servidor web Apache, el servidor de búsqueda Solr, la plataforma de mensajería Kafka o el proyecto sin duda más conocido de Apache, OpenOffice.

Cassandra se desarrolló con el objetivo de proporcionar una búsqueda eficiente para los usuarios de Facebook, facilitando la búsqueda de datos en sus bandejas de entrada. Para lograrlo, Facebook utilizó un clúster con más de 150 nodos individuales. No es sorprendente que Cassandra, con sus estructuras básicas similares a Amazon Dynamo y Google Bigtable, sea una elección popular entre los proveedores de redes sociales de gran escala. Algunos de los clientes destacados incluyen Twitter, Instagram, Spotify, así como los sitios web Digg y Reddit, que son portales de noticias sociales.

Cassandra utiliza un modelo de datos basado en columnas, lo que facilita a los desarrolladores la capacidad de agregar o eliminar columnas sin necesidad de reorganizar toda la base de datos. Además, está diseñado

específicamente para operar en clústeres de servidores, lo que garantiza una alta disponibilidad y un rendimiento escalable. También ofrece replicación automática de datos para asegurar la integridad y la tolerancia a fallos.

En resumen, las bases de datos NoSQL han revolucionado la forma en que almacenamos y gestionamos grandes volúmenes de datos hoy en día. Entre estas bases de datos, Cassandra destaca como un motor potente y confiable, capaz de manejar la escalabilidad y la distribución de datos de manera eficiente. En este trabajo de investigación, exploraremos la arquitectura en detalle mediante las características y el uso de Cassandra, proporcionándote las herramientas necesarias para enfrentar los desafíos del mundo de los datos en constante evolución.
Antes de profundizar en Cassandra, vamos a explorar brevemente las bases de datos NoSQL para tener una mejor comprensión del contexto.

¿Qué son las bases de datos NoSQL?

En el mundo digital actual, la cantidad de datos generados a diario es masiva. Desde información en redes sociales y transacciones financieras hasta registros médicos y demás, el volumen de datos ha crecido de manera exponencial. Esto plantea el desafío de almacenar, procesar y analizar eficientemente esta enorme cantidad de información, y aquí es donde entran en juego las bases de datos NoSQL y el concepto de Big Data.

La historia de las bases de datos NoSQL y el Big Data está íntimamente relacionada. A medida que la cantidad de datos crecía de forma exponencial, las bases de datos relacionales empezaron a mostrar limitaciones en términos de escalabilidad y rendimiento. Estando estas basadas en un modelo de datos estructurado, lo que dificultaba su adaptación a grandes volúmenes de información no estructurada.

Ante esta necesidad de manejar datos masivos, surgieron las bases de datos NoSQL, ofreciendo un enfoque más flexible y escalable para el almacenamiento y la gestión de datos. El término "NoSQL" se popularizó en la década de 2000, y su objetivo era proporcionar una variedad de modelos de datos, como documentos, clave-valor, grafos y columnas amplias.

Las bases de datos NoSQL permitían a las organizaciones manejar datos no estructurados o semiestructurados de manera eficiente. Proporcionando una mayor escalabilidad horizontal, lo que significa que los datos podrían distribuirse en múltiples servidores, lo que permitía un procesamiento paralelo y un manejo eficiente de grandes volúmenes de datos. Además, estos sistemas eran más flexibles en términos de esquema de datos, lo que permitía una adaptación más rápida a los cambios y nuevas fuentes de información.

A medida que las bases de datos NoSQL ganaban popularidad, surgieron nuevos desafíos asociados al manejo y análisis de grandes cantidades de datos. Este fenómeno se conoce como Big Data, que se refiere a conjuntos de datos extremadamente grandes y complejos que superan la capacidad de las herramientas tradicionales de procesamiento y análisis de datos.

El concepto de Big Data no solo se refiere a la velocidad de generación masiva de datos, sino también a la diversidad de fuentes y formatos. Para hacer frente a este desafío, se han desarrollado tecnologías y herramientas como sistemas de almacenamiento distribuido, plataformas de procesamiento en paralelo y algoritmos de análisis de datos a gran escala.

Finalmente podemos decir que las bases de datos NoSQL surgieron como una alternativa a las bases de datos relacionales tradicionales para abordar los desafíos de manejar grandes volúmenes de datos no estructurados. En el contexto del crecimiento del Big Data, Apache Cassandra se ha destacado como un motor de bases de datos NoSQL que ofrece una solución eficiente y escalable para el manejo y análisis de grandes volúmenes de datos. Su arquitectura distribuida y sus características avanzadas la convierten en una herramienta valiosa en el ámbito del Big Data.

Diferencias entre BBDD SQL y NoSQL

1. MODELO DE DATOS

Las bases de datos SQL siguen un modelo relacional, donde los datos se organizan en tablas con filas y columnas. Esto implica que los datos deben cumplir con una estructura predefinida. Las bases de datos NoSQL no tienen un esquema fijo y utilizan diferentes modelos de datos, como clave-valor, documentos, grafos o columnas amplias. Esto les brinda una mayor flexibilidad para manejar datos no estructurados o semiestructurados.

2. ESCALABILIDAD

Las bases de datos SQL suelen ser escalables verticalmente, lo que significa que se mejora su rendimiento al aumentar los recursos (como agregar más memoria o capacidad de procesamiento) en un único servidor. Por otro lado, las bases de datos NoSQL se diseñan para la escalabilidad horizontal, donde se pueden agregar más nodos o servidores al sistema para distribuir la carga y manejar grandes volúmenes de datos de manera más eficiente. Esto las hace especialmente adecuadas para aplicaciones que necesitan manejar el crecimiento exponencial de datos.

3. CONSULTAS Y TRANSACCIONES

Las bases de datos SQL son conocidas por su lenguaje de consulta estructurado y poderoso, el SQL, que permite realizar consultas complejas y realizar operaciones como *join* (unión) para combinar datos de diferentes tablas. Además, las bases de datos SQL son transaccionales, lo que significa que admiten transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) para garantizar la integridad de los datos en

operaciones de escritura. Por otro lado, las bases de datos NoSQL, debido a su enfoque en la escalabilidad y el rendimiento, pueden tener restricciones en cuanto a la complejidad de las consultas y la consistencia transaccional, aunque algunas bases de datos NoSQL, como Cassandra, ofrecen niveles configurables de consistencia.

4. CASOS DE USO

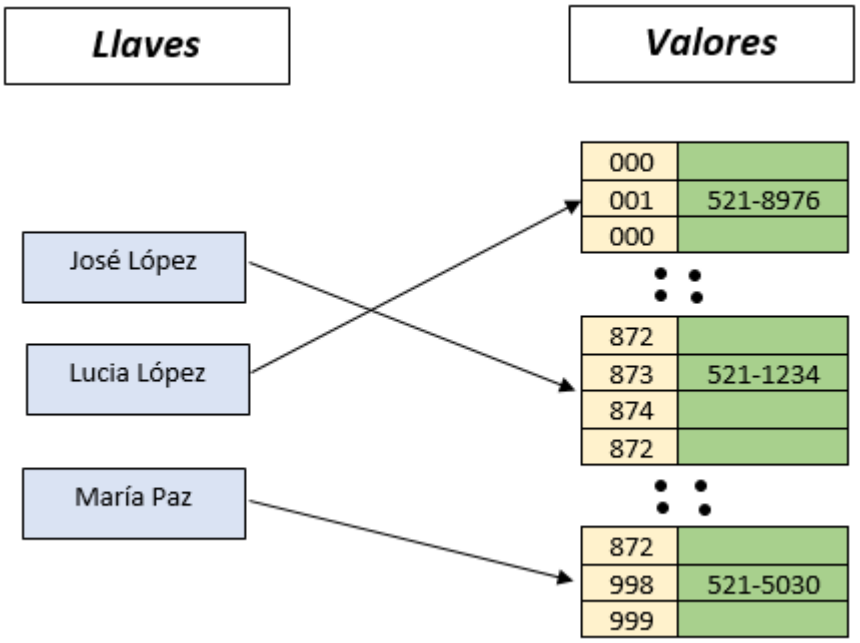
Las bases de datos SQL son adecuadas para aplicaciones que requieren estructura de datos definida, relaciones complejas y consultas flexibles. Son ampliamente utilizadas en aplicaciones empresariales, sistemas de gestión de bases de datos y aplicaciones que necesitan garantizar la integridad de los datos. Por otro lado, las bases de datos NoSQL son más adecuadas para aplicaciones que manejan grandes volúmenes de datos no estructurados o semiestructurados, como redes sociales, análisis de big data, sistemas de recomendación y aplicaciones en tiempo real.

Tipos de BBDD NoSQL

Existen varios tipos de bases de datos NoSQL que se han desarrollado para abordar diferentes necesidades y casos de uso. A continuación, se presentan algunos de los tipos más comunes de bases de datos NoSQL:

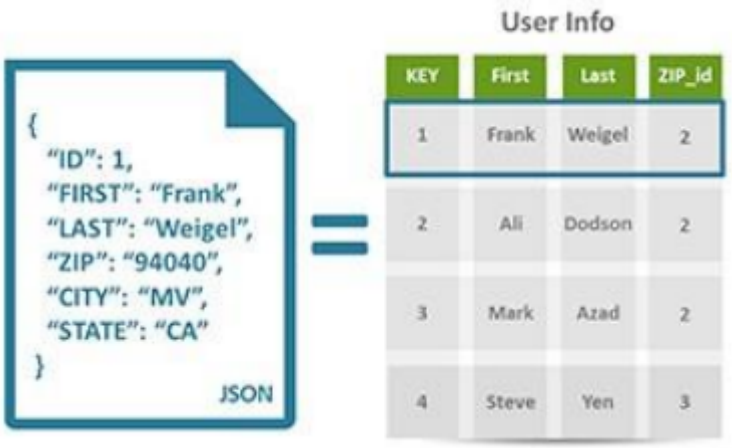
1. BASES DE DATOS DE CLAVE-VALOR

Este tipo de bases de datos almacena los datos en pares clave-valor, donde cada valor está asociado a una clave única. Las claves actúan como identificadores únicos para acceder a los valores almacenados. Son muy eficientes para operaciones de lectura y escritura rápidas, y son ampliamente utilizadas para caché, almacenamiento en memoria y aplicaciones que requieren alta velocidad de acceso a datos. Algunos ejemplos populares de bases de datos de clave-valor son Redis, Amazon, Riak y DynamoDB.



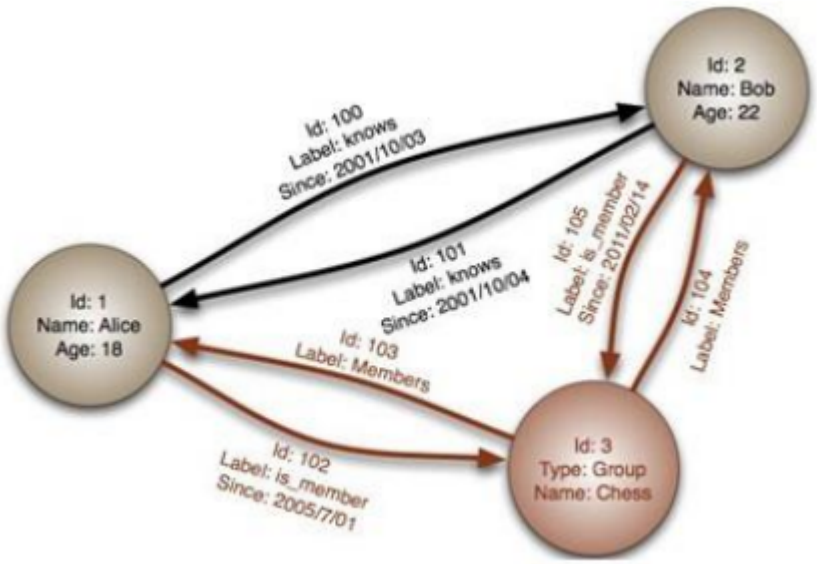
2. BASES DE DATOS DE DOCUMENTOS

Estas bases de datos almacenan y recuperan datos en formato de documentos, que pueden ser archivos JSON, XML o BSON. Cada documento contiene todos los datos necesarios para una entidad en particular, lo que facilita la manipulación y recuperación de datos completos. Las bases de datos de documentos son flexibles y escalables, ya que permiten agregar, modificar o eliminar campos en un documento sin afectar a los demás documentos en la colección. Son ideales para aplicaciones con datos semiestructurados o que requieren una flexibilidad en la estructura de los datos. MongoDB y CouchDB son ejemplos conocidos de bases de datos de documentos.



3. BASES DE DATOS DE GRAFOS

Los datos en las bases de datos de grafos se almacenan en forma de nodos interconectados y relaciones entre ellos. Este enfoque permite realizar consultas complejas y análisis de redes y relaciones entre los datos. Son especialmente útiles para aplicaciones que involucran análisis de redes sociales, recomendaciones personalizadas y navegación de datos altamente conectados. Ejemplos destacados de bases de datos de grafos incluyen Neo4j y OrientDB.



4. BASES DE DATOS DE COLUMNAS AMPLIAS

En este tipo de bases de datos, los datos se organizan en columnas en lugar de filas, lo que permite un acceso eficiente a conjuntos de datos masivos. Son adecuadas para aplicaciones que necesitan realizar consultas analíticas y agregaciones en grandes volúmenes de datos. Apache Cassandra y HBase son ejemplos populares de bases de datos de columnas amplias.

Motores de BBDD NoSQL

Dentro del mundo de las bases de datos NoSQL, existen varios motores o sistemas de gestión de bases de datos que se utilizan para implementar y administrar estos tipos de bases de datos. A continuación, se presentan algunos ejemplos destacados de motores de bases de datos NoSQL:

1. APACHE CASSANDRA

Cassandra es un motor de bases de datos NoSQL de columnas amplias y distribuido. Está diseñado para manejar grandes volúmenes de datos en un entorno escalable y de alta disponibilidad. Cassandra se basa en el modelo de datos de columnas amplias, lo que permite un rápido acceso a datos y una excelente capacidad de escritura. Además, ofrece replicación de datos en múltiples nodos, lo que garantiza la tolerancia a fallos y la recuperación ante desastres.

2. MONGODB

MongoDB es un popular motor de bases de datos NoSQL orientado a documentos. Almacena datos en formato JSON, lo que permite una estructura flexible y facilita el desarrollo de aplicaciones ágiles y escalables. MongoDB cuenta con características como índices, consultas ad hoc y replicación automática, lo que lo convierte en una opción popular para una amplia gama de casos de uso.

3. REDIS

Redis es un motor de bases de datos NoSQL de clave-valor de alto rendimiento. Es conocido por su velocidad y su capacidad para manejar cargas de trabajo intensivas de lectura y escritura. Redis almacena datos en memoria, lo que permite un acceso ultrarrápido a los datos, y también ofrece

persistencia en disco para garantizar la durabilidad de los datos. Es ampliamente utilizado para casos de uso como almacenamiento en caché, colas de mensajes y análisis en tiempo real.

4. NEO4J

Neo4j es un motor de bases de datos NoSQL de grafos, diseñado específicamente para almacenar y consultar datos altamente conectados, como redes sociales, sistemas de recomendación y análisis de relaciones. Permite el modelado y consulta de datos basados en nodos y relaciones, y ofrece un alto rendimiento en la navegación y búsqueda de grafos complejos. Neo4j es conocido por su capacidad de escalar y manejar grandes grafos de manera eficiente.

5. COUCHDB

Es compatible con una variedad de sistemas operativos, incluidos GNU/Linux, macOS y Windows. CouchDB está escrito en Erlang y utiliza la plataforma de virtualización de máquina Erlang (BEAM) para su funcionamiento. De fácil uso y replicación de datos en múltiples servidores.

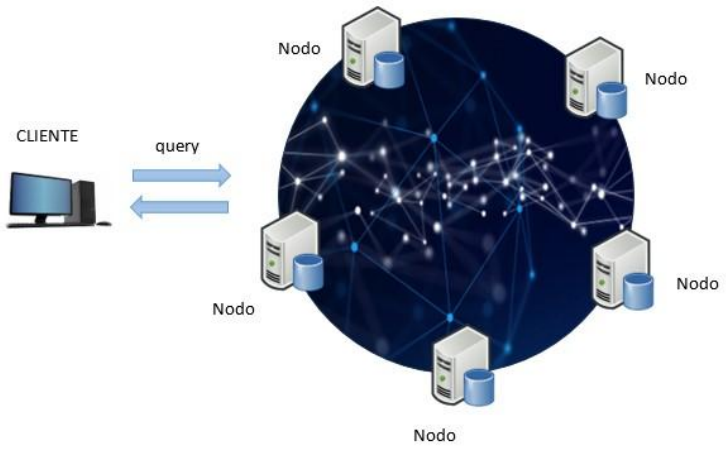
Estos son solo algunos ejemplos de motores de bases de datos NoSQL, cada uno con sus características y fortalezas particulares. La elección del motor de base de datos NoSQL dependerá de los requisitos específicos del proyecto, la estructura de datos y las necesidades de escalabilidad y rendimiento. Es importante evaluar cuidadosamente cada motor para seleccionar el más adecuado para el caso de uso específico.

Como características más importantes cabe destacar el uso de Restful HTTP API como interfaz y JavaScript como principal lenguaje de interacción. Para el almacenamiento de los datos se utilizan archivos JSON. Permite la creación de vistas, que son el mecanismo que permite la combinación de documentos para retornar valores de varios documentos, es decir, CouchDB permite la realización de las operaciones JOIN típicas de SQL.

Cassandra

Cassandra es una base de datos distribuida tipo NoSQL de código abierto, cuyo diseño y nombre están inspirados en la sacerdotisa Cassandra de la mitología griega. El logo de Cassandra, que representa un ojo, simboliza la capacidad de la base de datos para proporcionar una visión profunda y detallada de los datos almacenados.
En cuanto a la distribución de los datos, Cassandra utiliza una arquitectura distribuida basada en un clúster de nodos. Esta arquitectura peer-to-peer (P2P) permite que los nodos se comporten como iguales, actuando como clientes y servidores entre sí.

La arquitectura de Cassandra es escalable, lo que significa que se puede agregar fácilmente un nuevo nodo para aumentar el espacio de almacenamiento y la capacidad de procesamiento. Los datos se distribuyen en el clúster, representado como un anillo o círculo, dividido en rangos iguales al número de nodos. Cada nodo es responsable de uno o más rangos de datos.



Para la distribución de datos, Cassandra utiliza tokens. Cada nodo tiene asignado un rango determinado de tokens, que se utilizan para buscar y ubicar los datos. Al buscar una fila de datos en Cassandra, se calcula el token asociado y se determina el nodo al que pertenece. Los tokens están ordenados en al anillo, donde el último token es considerado el predecesor del primer nodo.

Cuando se agregan o eliminan nodos en el clúster, los tokens asignados a cada nodo deben redistribuirse. Esto nos garantiza que los datos estén equilibrados y distribuidos de manera adecuada en el clúster.
Cassandra también ofrece la replicación del dato a través de distintos nodos para garantizar la tolerancia a fallos y la alta disponibilidad. El factor de replicación (RF) determina el número de nodos utilizados para almacenar cada dato.

Cassandra usa dos estrategias para la replicación de datos:

- SimpleStrategy es la estrategia por defecto en Cassandra, escribe los datos a los nodos posteriores a lo largo del anillo. Cassandra tiene guardado un mapa de red para que cada nodo sepa cual es el siguiente en el anillo. Esta estrategia se implementa en un centro de datos un solo cluster.

- NetworkTopologyStrategy esta estrategia nos sirve cuando queremos implementar el cluster en varios centros de datos, teniendo en cuenta cuántas réplicas se desean en cada uno de estos. Y además se

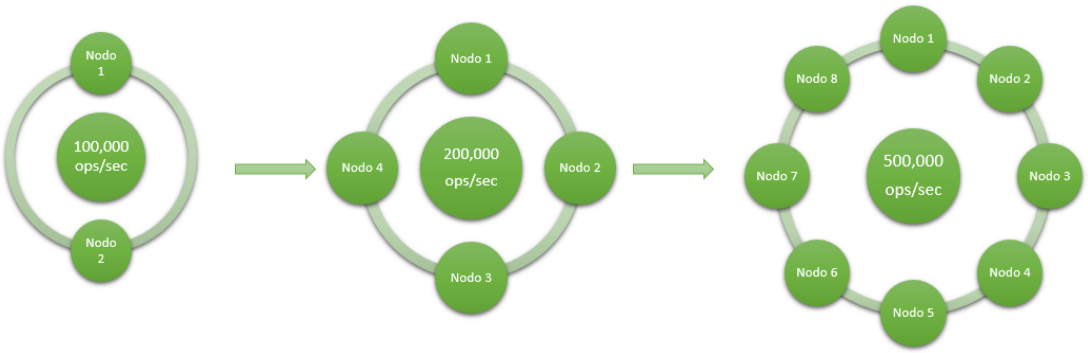
puede controlar la distribución de las réplicas en función de la topología de red y la ubicación geográfica.

Ventajas y Desventajas de los sistemas NoSQL, con énfasis en Apache Cassandra

Ventajas

1. ESCALABILIDAD

Es una ventaja que tiene el sistema para poder procesar mayor información, ya que a medida que crece se debe pensar en la distribución de la carga de trabajo sin perder rendimiento o sin tener caídas en el servidor. La idea es aumentar el número de nodos o clusters para poder manejar así grandes volúmenes de datos y tener un respaldo por posibles fallas mediante la replicación de estos datos, al aumento de nodos se lo llama *escalabilidad horizontal*. Cassandra está diseñado para funcionar en clusters distribuidos, lo que permite una escalabilidad casi lineal.



2. FLEXIBILIDAD PARA ALMACENAR DATOS

En el mundo de las bases de datos NoSQL, Cassandra garantiza flexibilidad y adaptación al momento de almacenar datos, ya que no requiere de un esquema fijo o predefinido. Permite una mayor optimización al agregar, modificar o eliminar campos en los datos almacenados. Además, tenemos un mayor alcance y rendimiento para las operaciones de base de datos mejorando así la eficiencia del flujo de trabajo, con una estructura dinámica, evitando realizar cambios en todo el esquema cuando se agregan nuevos campos.

3. RENDIMIENTO OPTIMIZADO

Las bases de datos NoSQL ofrecen un rendimiento optimizado a los usuarios al ser una base de datos distribuida, de código abierto y de alto rendimiento. Es una herramienta intuitiva de fácil uso y su robustez y escalabilidad permiten a los usuarios manejar grandes volúmenes de datos.

4. ALTA DISPONIBILIDAD

Esta característica de Cassandra permite a los usuarios mantener sus servicios en línea sin interrupción, incluso en caso de fallas. También, está diseñado para resistir la pérdida de datos y la caída de los nodos sin afectar la disponibilidad. Esto es gracias a que Cassandra no sigue un patrón maestro servidor o también llamado “esclavo”, sino que es peer-to-peer, garantizando que si se cae un nodo, el servicio puede seguir funcionando.

5. NO GENERA CUELLOS DE BOTELLA

El principal problema de los sistemas SQL es que necesitan transcribir cada sentencia para poder ser ejecutada, y cada sentencia compleja requiere además de un nivel de ejecución aún más complejo, lo que constituye un punto de entrada en común, que ante muchas peticiones puede ralentizar el sistema.

6.GESTIÓN AUTOMÁTICA DE REPLICACIÓN

Como ya se mencionó en otras ventajas Cassandra ofrece una replicación automática de datos entre los nodos de la base de datos, lo que proporciona un alto nivel de disponibilidad de los datos y tolerancia a fallos.

7. INTEGRACIÓN CON OTRAS TECNOLOGÍAS DE CÓDIGO ABIERTO

Cassandra se integra bien con otras tecnologías de código abierto y ecosistemas de Big Data. Puede funcionar por ejemplo junto con el framework Apache Spark o entornos de trabajo como Apache Hadoop y otras tecnologías populares para análisis y procesamiento de datos, lo que permite casos de uso avanzados, como análisis en tiempo real y aplicaciones de aprendizaje automático.

8. DOCUMENTACION, COMUNIDAD Y SOPORTE

Apache Cassandra cuenta con una comunidad activa, con amplia documentación, tutoriales y recursos disponibles. Además, hay proveedores y empresas que brindan soporte y servicios profesionales para implementaciones de Cassandra.

Desventajas:

1. MENOR SOPORTE PARA CONSULTAS COMPLEJAS

A menudo no se admiten consultas que involucren muchas tablas o relaciones, limitando su uso para poder ejecutar procesos más complejos. Además, no se pueden realizar operaciones como actualizar o eliminar datos en varias tablas al mismo tiempo. Por ende, Cassandra no es una buena opción para proyectos que requieran una manipulación de datos avanzada, ya que son usadas para consultas simples y rápidas en grandes conjuntos de datos.

2. FALTA DE HERRAMIENTAS DE ADMINISTRACIÓN

Hay herramientas que son esenciales en una base de datos, tanto para configurar como para administrar. En Cassandra estas herramientas son limitadas dificultando la realización de tareas como por ejemplo la replicación de datos y posiblemente conlleve a la dificultad para la escalabilidad de la base de datos, limitando su alcance solo a proyectos de menor escala.

3. MENOR CONSISTENCIA TRANSACCIONAL

En situaciones de alta concurrencia, es posible que se produzcan lecturas o escrituras inconsistentes en diferentes nodos del clúster. Sin embargo, Cassandra ofrece niveles de consistencia configurables, lo que permite equilibrar la coherencia y el rendimiento según las necesidades del proyecto.

4. RENDIMIENTO LIMITADO

Cassandra no ofrece soporte para transacciones complejas ni para consultas avanzadas. Esto significa que los usuarios tienen limitaciones en cuanto a la cantidad de datos que pueden recuperar de la base de datos en una sola consulta. Además, la arquitectura de Cassandra tiene una escalabilidad vertical limitada, lo que significa que los usuarios no pueden

aprovechar el aumento del rendimiento de la base de datos al añadir nuevos servidores.

5. ALTO COSTO DE MANTENIMIENTO

El alto costo de mantenimiento es una desventaja importante para el uso de Cassandra ya que esta característica podría llegar a ser una barrera para aquellos que quieran usar esta herramienta, ya que se requiere una inversión considerable en el tiempo para llevar a cabo el mantenimiento y asegurar el funcionamiento óptimo.

6. ES NECESARIO REALIZAR UN MODELADO DE DATOS CUIDADOSO

A fines de obtener un aprovechamiento máximo de Cassandra, es necesario realizar un modelado de datos cuidadoso y considerar los patrones de acceso a los datos. Esto puede requerir un conocimiento más profundo de cómo funciona Cassandra y cómo diseñar el esquema de datos correctamente. La falta de un esquema rígido puede hacer que sea más fácil cometer errores en el diseño del modelo.

7. CURVA DE APRENDIZAJE

Requiere tiempo, dedicación y esfuerzo aprender a utilizar y administrar Cassandra de manera eficaz especialmente cuando no se cuenta con mucha experiencia con bases de datos NoSQL distribuidas. Su curva de aprendizaje puede ser muy elevada para aquellos con más experiencia de bases de datos relacionales tradicionales.

Teorema CAP

Es importante hablar sobre el teorema CAP el cual establece que es imposible para un sistema distribuido de datos cumplir en simultáneo con tres propiedades:

1. DISPONIBILIDAD

Todos los nodos de la base de datos están disponibles para recibir y responder a las solicitudes de los clientes en todo momento.

2. TOLERANCIA A PARTICIONES

Indica la capacidad del sistema para seguir funcionando incluso si algún nodo de la base de datos están inaccesibles debido a una partición de red.

3. CONSISTENCIA

Todos los nodos de la base de datos ven los mismos datos al mismo tiempo. Tenemos tres modos de consistencias:

Bajo: existe el riesgo de que el primer nodo disponible en el sistema pueda devolver un dato antiguo. Sin embargo, la ventaja es que se obtiene una respuesta más rápida, ya que no se espera a que todos los nodos estén de acuerdo antes de devolver el resultado.

Medio: se reduce la probabilidad de devolver un dato antiguo. Cassandra busca llegar a un acuerdo entre los nodos antes de devolver un resultado. Esto implica un poco más de tiempo para obtener el dato, pero se reduce el riesgo de obtener información desactualizada.

Modo Alto: Cassandra espera a que todos los nodos que tienen el dato respondan antes de devolver el resultado. Esto garantiza que se obtenga el dato más actualizado, pero a costa de sacrificar la velocidad de la respuesta, ya que se necesita esperar a que todos los nodos respondan.

Cuando tenemos una partición de red el sistema distribuido debe elegir entre mantener la consistencia o la disponibilidad. Osea que el sistema debe optar por responder a las solicitudes de los clientes o mantener la consistencia de los datos.

En resumen según el teorema, Cassandra es una base de datos AP(Availability and Partition tolerance) donde el sistema se enfoca principalmente en la disponibilidad y tolerancia a partición, para que pueda seguir operando incluso si hay alguna falla en algún nodo del cluster, o

inaccesibilidad por partición de red, esto es posible gracias a la replicación de datos en los distintos nodos, garantizando la información allí guardada.
La replicación de datos tiene un inconveniente en cuanto a la disponibilidad de almacenamiento, ya que más replicación usemos menos memoria tendremos disponible.

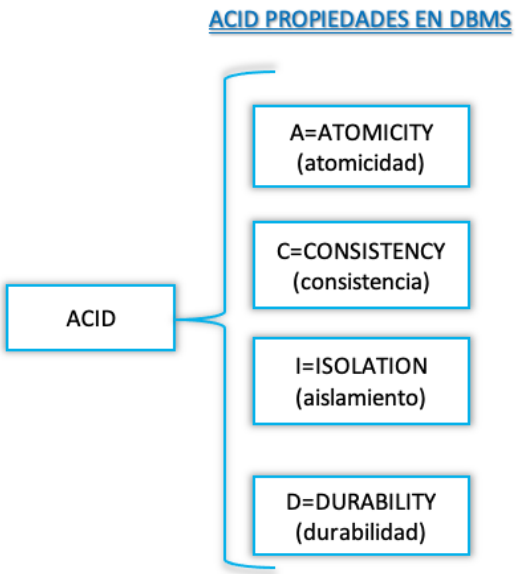
Otra de las desventajas al enfocarse en las propiedades AP. Da lugar a una menor consistencia en la base de datos, ya que los nodos pueden no tener los mismos datos en el mismo momento, en lugar de eso Cassandra utiliza un modelo de consistencia basado en el tiempo para poder lograr eventualmente una consistencia completa.

Características de Apache Cassandra

Algunas de las características fundamentales de Cassandra son:

1. Altamente escalable
2. Altamente flexible ya que soporta datos estructurados, semi-estructurados y no estructurados.
3. Tiene soporte para transacciones ya que sigue las propiedades ACID*.

   Una transacción es una unidad lógica de trabajo que accede y posiblemente modifica el contenido de una base de datos. Las transacciones acceden a los datos mediante operaciones de lectura y escritura.



ATOMICIDAD

Quiere decir que la transacción completa se realiza de una sola vez o no se realiza. Implica dos operaciones:

a) Abortar - Los cambios efectuados en la BD no son visibles cuando se cancela una transacción.
b) Confirmar - Los cambios efectuados en la BD son visibles cuando se confirma una transacción.

CONSISTENCIA
Significa que se deben mantener las restricciones de integridad para que la base de datos posea consistencia antes y después de la transacción.

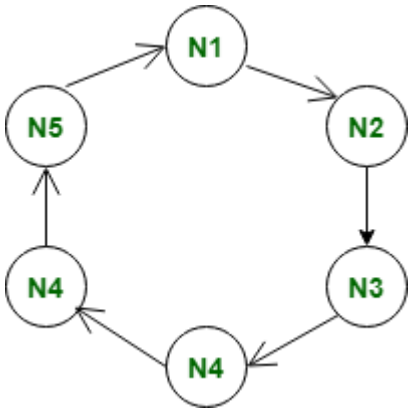
AISLAMIENTO
Asegura que se puedan realizar varias transacciones al mismo tiempo sin que se produzca una incoherencia en el estado de la BD.

DURABILIDAD
Asegura que una vez que la transacción haya finalizado su ejecución, las actualizaciones y modificaciones a la BD se almacenen y escriban en el disco.

Las propiedades ACID proporcionan un marco para garantizar la coherencia, la integridad y confiabilidad de los datos. Aseguran que las transacciones se ejecuten de manera confiable y consistente, incluso si hubiese fallas de sistema.

4. Es altamente disponible y tolerante a fallas (La tolerancia a fallas es el proceso de funcionamiento de un sistema de manera adecuada a pesar de la ocurrencia de fallas de sistema).

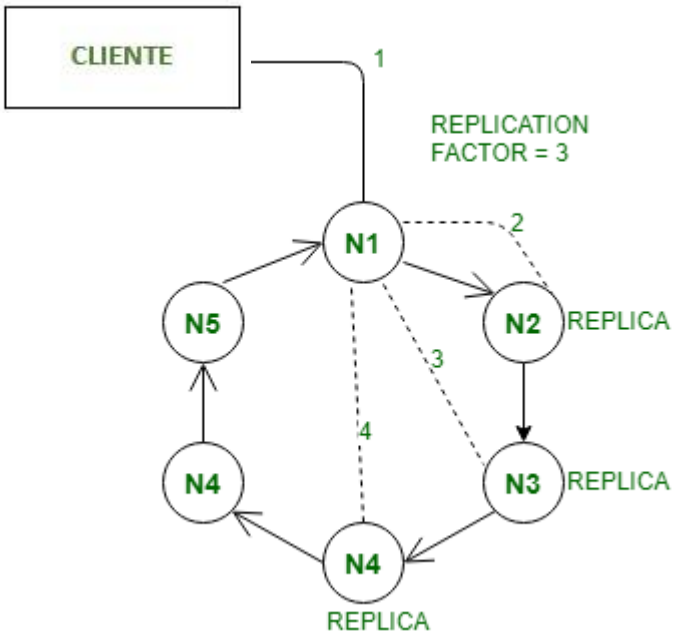
5. Es Open-Source
En Cassandra, no existe una arquitectura maestro-cliente, sino una comunicación peer-to-peer donde se puede crear varias copias de datos al momento de la creación del espacio de claves.
Para esto, se debe definir la estrategia de replicación y factor de replicación (RF) para crear múltiples copias de datos.



Un ejemplo de esto podría ser el siguiente:

```
CREATE KEYSPACE Example
WITH replication = {'class': 'NetworkTopologyStrategy',
                    'replication_factor': '3'};
```

En este caso, estamos creando 3 copias de datos en varios nodos en sentido de las agujas de reloj.



Tipos de consulta CQL

En Cassandra, se pueden realizar varias consultas para manipular y recuperar datos. Las consultas más comunes que se pueden realizar son:

1. Consultas básicas de lectura/escritura: Se puede insertar y recuperar datos utilizando el lenguaje de consulta de Cassandra (CQL). Por ejemplo:

- INSERT INTO <tabla> (<columnas>) VALUES (<valores>): Inserta datos en una tabla.
- SELECT <columnas> FROM <tabla> WHERE <condiciones>: Recupera datos de una tabla.

2. Consultas por clave primaria: Cassandra está diseñada para permitir un acceso eficiente a los datos utilizando la clave primaria. Puedes realizar consultas basadas en la clave primaria utilizando la cláusula WHERE. Por ejemplo:

- SELECT <columnas> FROM <tabla> WHERE <clave primaria> = <valor>: Recupera filas que coinciden con un valor de clave primaria específico.
- SELECT <columnas> FROM <tabla> WHERE <clave primaria> IN (<valores>): Recupera filas que coinciden con varios valores de clave primaria.

3. Consultas de rango: Cassandra permite realizar consultas de rango utilizando la cláusula WHERE con los operadores de comparación. Por ejemplo:

- SELECT <columnas> FROM <tabla> WHERE <columna> > <valor>: Recupera filas donde una columna sea mayor que un valor dado.
- SELECT <columnas> FROM <tabla> WHERE <columna> BETWEEN <valor1> AND <valor2>: Recupera filas donde una columna esté en un rango específico.

4. Consultas de agregación: Puedes realizar consultas de agregación para calcular sumas, promedios, máximos, mínimos, etc. en una columna o grupo de columnas. Por ejemplo:

- SELECT COUNT(*) FROM <tabla>: Devuelve el recuento total de filas en una tabla.

- SELECT AVG(<columna>) FROM <tabla>: Calcula el promedio de los valores en una columna.

5. Consultas por índices secundarios: Cassandra permite crear índices secundarios en columnas específicas para mejorar las consultas de búsqueda. Puedes realizar consultas utilizando índices secundarios. Por ejemplo:

- CREATE INDEX <nombre_indice> ON <tabla> (<columna>): Crea un índice secundario en una columna.
- SELECT <columnas> FROM <tabla> WHERE <columna> = <valor>: Realiza una búsqueda utilizando un índice secundario.

6. Creación de una tabla: define la estructura de la tabla y sus columnas, asignando un nombre a la tabla y especificando el nombre de las columnas conjunto con los tipo de datos que va a contener. Además, cada tabla tendrá una clave primaria para identificar de manera única cada fila de datos.

-CREATE TABLE <tabla> (<columnas> PRIMARY KEY (<clave_primaria>));

Ejemplos de consultas CQL

Para poder interactuar con Cassandra a través de CQL (Cassandra Query Language) debemos utilizar CQL Shell (cqlsh) el cual es un shell de línea de comandos.

Para poder realizar una consulta CQL se deben seguir los siguientes pasos:

PASO 1. Se debe crear un espacio de claves.

```
CREATE KEYSPACE Emp
WITH replication = {'class': 'SimpleStrategy',
                    'replication_factor': '1'};
```

PASO 2.

```
Syntax:
USE keyspace-name
USE Emp;
```

PASO 3. Se debe crear una tabla utilizando la siguiente consulta CQL.

```
Example:
CREATE TABLE Emp_table (
    name text PRIMARY KEY,
    Emp_id int,
    Emp_city text,
    Emp_email text,
    );
```

PASO 4. Para insertar en Emp_table, se debe utilizar la siguiente consulta CQL.

```
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('ashish', 1001, 'Delhi', 'ashish05.rana05@gmail.com');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('Ashish Gupta', 1001, 'Bangalore', 'ashish@gmail.com');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('amit ', 1002, 'noida', 'abc@gmail.com');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('dhruv', 1003, 'pune', 'xyz@gmail.com');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('shivang', 1004, 'mumbai', 'test@gmail.com');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('aayush', 1005, 'gurugram', 'cass_write@gmail.com');
Insert into Emp_table(name, Emp_id, Emp_city, Emp_email)
  VALUES ('bhagyesh', 1006, 'chandigar', 'welcome@gmail.com');
```

PASO 5. Para leer datos, se debe utilizar la siguiente consulta CQL.

```
SELECT * FROM Emp_table;
```

Cassandra y sus Componentes

-Clúster: cuando se habla de clúster nos referimos a un centro de datos que contiene varios nodos de Cassandra. También hace referencia al conjunto de servidores que trabajan juntos para almacenar y administrar los datos.

-Datacenter: es una colección de nodos relacionados dentro de un clúster. Puede haber distintos datacenter en un clúster, y cada uno puede estar ubicado físicamente en diferentes lugares geográficos.

-Nodo: un nodo es donde se almacenan los datos en Cassandra. Cada nodo es responsable de un subconjunto de los datos totales en el clúster. Estos se comunican entre sí para garantizar la replicación de los datos.

-Commit log: es un archivo donde se registra la información sobre los cambios en los datos, funciona como una especie de copia de seguridad/respaldo en caso de que haya una falla con el sistema. Permite recuperar los datos y mantener la integridad de los mismos.

-MemTable: es una estructura de almacenamiento en memoria que contiene los datos que aún no se han escrito en los archivos de datos en disco (SSTables). Permite un acceso rápido a los datos antes de que se escriban en disco.

- SSTable: es un archivo en disco que almacena los datos escritos. Una vez que se crea un archivo SSTable no se puede modificar. Cada SSTable contiene una porción de los datos totales en el clúster.

-About internode communications: es el protocolo de comunicación peer-to-peer utilizado por los nodos para descubrir y compartir información sobre la ubicación y el estado de otros nodos en el clúster. Esto permite que los nodos se mantengan actualizados sobre la configuración y los cambios en el clúster.

-Partitioner: Es el componente que determina cómo se distribuyen los datos entre los nodos en el clúster. Define la función de hash que se utiliza para asignar las filas de datos a los nodos correspondientes. Así es como se asegura una distribución equilibrada de los datos en el cluster.

-Replica placement strategy: es la estrategia utilizada para almacenar copias de los mismos datos en diferentes nodos. Esto garantiza la

accesibilidad a los datos y la tolerancia a fallos. Puede especificar cuántas réplicas se deben mantener y dónde deben colocarse en el clúster.

- *Snitch:* Define la estructura de red del clúster de Cassandra. Utilizando estrategias de replicación para colocar las réplicas y dirigir las consultas de forma eficiente. Esta herramienta permite a Cassandra conocer la topología de red y utilizar esa información para enrutar solicitudes y distribuir réplicas de datos de manera adecuada. Además, distribuye las réplicas en ubicaciones estratégicas para mejorar la disponibilidad y la resistencia a fallas del sistema.

Cassandra Proceso de Escritura y Lectura

El ***proceso de escritura*** en Cassandra se realiza de la siguiente manera:

1. El usuario hace una solicitud de escritura, la cual se envía a la MemTable, está temporalmente almacena la escritura en la memoria RAM para cada familia de columnas.

2. Luego, la escritura se realiza en cada nodo. El commit log guarda la actividad y la información de escritura para garantizar la durabilidad de los datos. Estos datos son guardados en el disco.

3. Cuando la MemTable se queda sin espacio, se realiza una operación llamada "Flush". En donde, los datos se eliminan de la MemTable, liberando así memoria.

4. Los datos eliminados en la etapa de Flush se escriben en disco, en archivos llamados SSTables. Estos archivos al ser inmutables, no se vuelven a escribir después de almacenar los datos.

5. Una vez que todos los datos están escritos en las SSTables, se intenta reducir el espacio que ocupan. Esto se logra eliminando las SSTables obsoletas que han sido escritas con datos actualizados. Sin embargo, los datos actualizados se escriben en una nueva SSTable en lugar de reescribirlos en la misma. Este proceso se lo llama compactación.

Las escrituras son mucho más costosas en el momento en que se escriben en las SSTables. Sin embargo, si los datos se encuentran en la MemTable en la memoria, la escritura y el acceso son más rápidos y menos costosos.

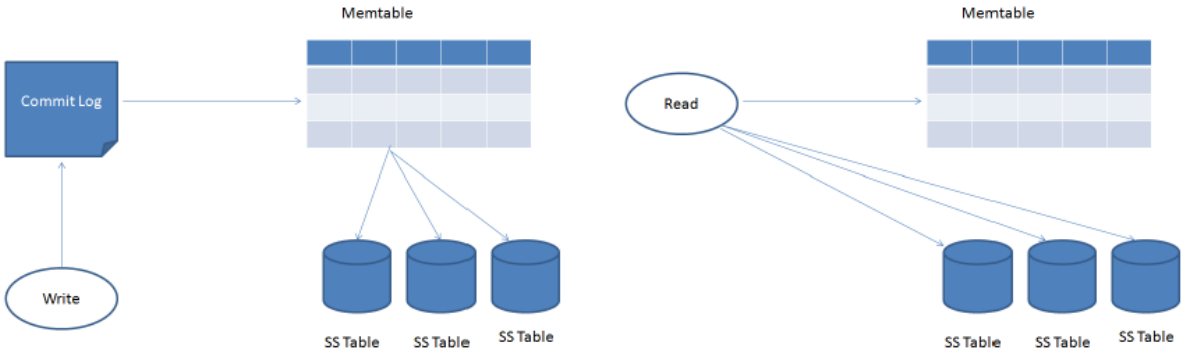
Cassandra puede reescribir datos en filas, lo que significa que no realiza una búsqueda para verificar si los datos existen antes de la escritura.
Los datos en Cassandra pueden tener una fecha de caducidad (LLTS), a partir de la cual se marcan como eliminados. Esto se utiliza como un recurso adicional durante la compactación.

El ***proceso de lectura*** en Cassandra varía según la ubicación de la información:
- Si los datos se encuentran en la MemTable, se recuperan de manera rápida, eficiente, y se envían.
- Si los datos no están en la MemTable, Cassandra accede a las SSTables para buscar la información.
Existen tres tipos de lecturas en Cassandra:

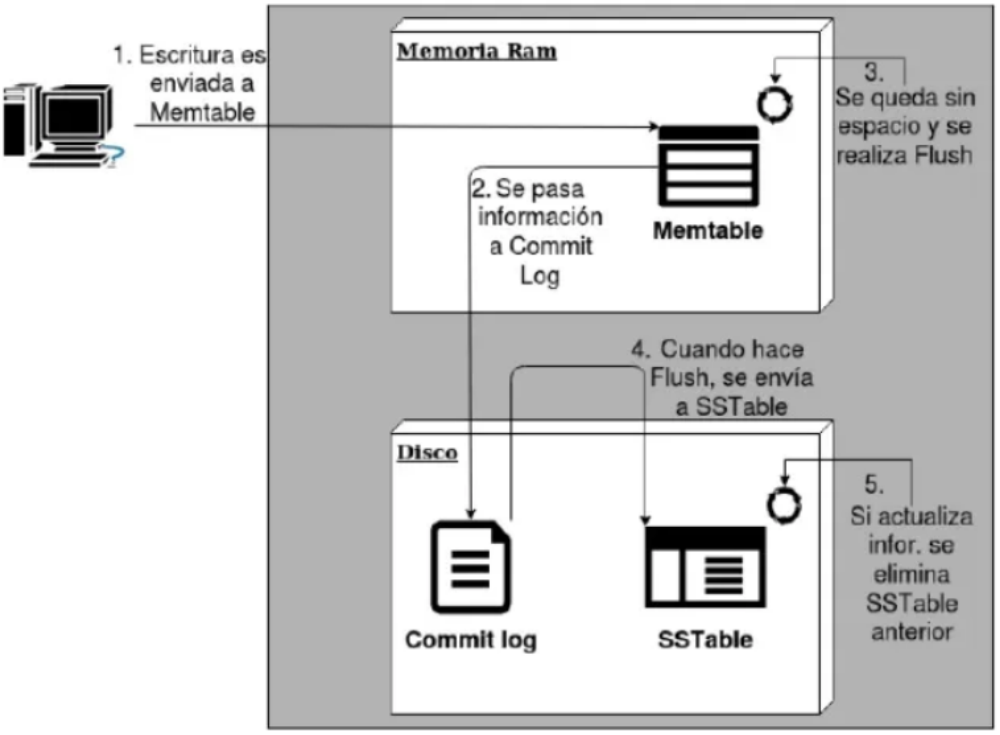
1.***Direct read request*** **(solicitud de lectura directa):** El nodo coordinador se comunica con un nodo que contiene la réplica solicitada y recupera los datos.

2.***Digest request*** **(solicitud de resumen):** El nodo coordinador se comunica con varios nodos que tienen réplicas y verifica la consistencia de los datos retornados por el nodo contactado en la solicitud de lectura directa.

3.***Background read repair request*** **(solicitud de reparación de lectura en segundo plano):** Si hay inconsistencias entre las réplicas, las réplicas con un timestamp más antiguo se sobrescriben con los datos de la réplica más reciente.



(a) Almacenamiento (b) Proceso de lectura


1. Escritura es enviada a Memtable
Memoria Ram
3. Se queda sin espacio y se realiza Flush
2. Se pasa información a Commit Log
Memtable
4. Cuando hace Flush, se envía a SSTable
Disco
5. Si actualiza infor. se elimina SSTable anterior
Commit log
SSTable

Conclusión

En este trabajo de investigación, hemos explorado las bases de datos NoSQL, el Big Data y específicamente el motor de base de datos Cassandra. Hemos comprendido su importancia y relevancia en el manejo de grandes volúmenes de datos.

Comenzamos estudiando las bases de datos NoSQL y las diferencias con las bases de datos SQL tradicionales. Conocimos los diferentes tipos de bases de datos NoSQL, como las basadas en documentos, clave-valor, grafos y columnas amplias.

Luego, nos enfocamos en Cassandra, un motor de base de datos distribuido y escalable. Analizamos sus ventajas y desventajas, así como el Teorema CAP que establece las limitaciones en sistemas distribuidos.

Exploramos las características específicas de Cassandra, como su modelo de datos basado en columnas y su capacidad para consultas eficientes. También examinamos los componentes principales de Cassandra y su proceso de escritura y lectura de datos en un entorno distribuido.

En conclusión, hemos aprendido que las bases de datos NoSQL, el Big Data y Cassandra son fundamentales en el manejo de grandes volúmenes de datos. Estos temas representan áreas emocionantes y en constante evolución en la tecnología. A medida que avanzamos en este campo, reconocemos la importancia de la innovación y esperamos seguir aprendiendo y contribuyendo al avance de este campo dinámico.

OPINIONES DE PORQUE USARÍAMOS CASSANDRA

Campott, Daiana:
Elegiría Cassandra por su capacidad de escalabilidad horizontal, lo que nos permite manejar grandes volúmenes de datos. Además, destaco su tolerancia a fallos gracias a la replicación de datos en los nodos, lo que la convierte en una base de datos confiable para información importante. En resumen, Cassandra me parece una opción eficiente y de alto rendimiento para proyectos de empresas que requieran estas características.

Carlini Gianluca:
Utilizaría Cassandra en varias situaciones. Su diseño distribuido y tolerante a fallas la convierte en una opción sólida para aplicaciones que requieren alta disponibilidad y escalabilidad horizontal. Cassandra también es ideal para casos de uso con cargas de escritura intensivas y consultas rápidas, gracias a su modelo de datos orientado a columnas y su capacidad para manejar grandes volúmenes de datos. Finalmente, su capacidad para funcionar en múltiples centros de datos la hace adecuada para aplicaciones globales por lo que puede ser más conveniente optar por este tipo de base de datos para sistemas que requieran NoSQL.

Aldana A. Nesteruk Bonazzola:
Para un usuario común y corriente, el uso de Cassandra puede no ser la opción más adecuada. Cassandra está diseñada principalmente para casos de uso en entornos empresariales y aplicaciones que manejan grandes volúmenes de datos y requieren alta escalabilidad y rendimiento.
En resumen, para un usuario común y corriente sin necesidades específicas de escalabilidad y rendimiento masivo, otras bases de datos más sencillas y fáciles de usar serían más adecuadas y menos complicadas de implementar y mantener.

Gibon, Florencia Ayelen
Cassandra es una opción eficiente y de alto rendimiento para proyectos empresariales que requieren escalabilidad, tolerancia a fallos y manejo de grandes volúmenes de datos. Aun así, es importante evaluar las necesidades y capacidades del sistema de datos a usar antes de decidir utilizar Cassandra, ya que puede o no, ser la opción más adecuada para usuarios que no tengan manejo de gran volumen de información y sin requerimientos específicos de escalabilidad.

Paladini, Nicolás
Cassandra sin dudas es una una elección sólida para aplicaciones que necesitan manejar grandes volúmenes de datos, operaciones intensivas y requieren alta disponibilidad y escalabilidad sin comprometer el rendimiento. Tal es el ejemplo de las empresas que utilizar este tipo de base de datos como Netflix e Instagram que justifican su implementación dado el volumen de datos que manejan, pero en los casos de bases de datos de menor escala debemos considerar lo costoso de su implementación y su empinada curva de aprendizaje suponiendo que quien lo implementa no tiene experiencia previa suficiente.

BIBLIOGRAFÍA

https://hostingdata.co.uk/nosql-database/

https://web.archive.org/web/20100307115031/http://barrapunto.com/articles/09/11/02/1227257.shtml?tid=136

https://web.archive.org/web/20110217092700/http://www.nosql.es/blog/

https://www.proscont.com/ventajas-y-desventajas-de-cassandra/

https://geekflare.com/es/apache-cassandra/

https://www.geeksforgeeks.org/five-main-benefits-of-apache-cassandra/amp/?ref=rp

https://www.geeksforgeeks.org/apache-cassandra-nosql-database/

https://www.geeksforgeeks.org/cassandra-nosql-database/

https://cassandra.apache.org/doc/latest/cassandra/architecture/overview.html

https://1938.com.es/cassandra

Cassandra: La mejor base de datos para la IoT

https://abdatum.com/tecnologia/cassandra

https://www.paradigmadigital.com/dev/cassandra-la-dama-de-las-bases-de-datos-nosql/

https://aprenderbigdata.com/introduccion-apache-cassandra/

https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/architecture/archDataDistributeReplication.html

https://cassandra.apache.org/doc/latest/cassandra/architecture/snitch.html

https://medium.com/@jarrietaalfaro/cassandra-funcionamiento-y-principios-926d5bebe576

https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/apache-cassandra/