Introducción

Apache Cassandra - Bases de datos II

Alberto Díaz Álvarez (<alberto.díaz@upm.es>)
Departamento de Sistemas Informáticos
Escuela Técnica superior de Ingeniería de Sistemas Informáticos

En este tema hablaremos de las bases de datos clave-valor

- Sus propiedades, escalabilidad, indexación, etcétera
- Nos centraremos en Apache Cassandra, aunque sea un modelo clave-valor híbrido
    - Híbrido porque incluye el concepto de columnas

Su *paper* de referencia:

- Lakshman, A., & Malik, P. (2010). **Cassandra: a decentralized structured storage system**. *ACM SIGOPS* operating systems review, 44(2), 35-40.

Bases de datos clave-valor

¿Qué es una base de datos clave-valor?

Es un sistema que almacena valores indexados por claves

- Vamos, como una tabla *hash*, pero a lo bestia
- Permite almacenar datos estructurados y no estructurados

*"Apache Cassandra is an **open source**, **distributed**, **decentralized**, **elastically scalable**, **highly available**, **fault-tolerant**, **tunable** and **consistent** database that bases its distribution design on Amazon's Dynamo and its data model on Google's Bigtable. Created at Facebook, it is now used at some of the most popular sites on the web."*

- Cassandra: The definitive guide (J. Carpenter y E. Hewitt -

Sobre Apache Cassandra

Sistema de gestión de bases de datos **NoSQL distribuido**

- Arrancado por Facebook y posteriormente cedido a la Apache Foundation
- Gratuito y de código abierto (licenciado bajo la Apache License 2.0)
- Desarrollado en el lenguaje de programación Java y multiplataforma
- Diseñado para manejar grandes cantidades de datos entre muchos nodos
- Soporta clústeres distribuidos en diferentes clústeres
- Además con replicación asíncrona sin nodo maestro
- Esto permite operaciones de baja latencia
- Web: https://cassandra.apache.org
- Algunos sitios que usan Cassandra: Netflix, Spotify y Uber

Su sitio en la familia NoSQL

Las bases de datos NoSQL suelen estar orientadas a casos muy específicos

- Vamos a compararla con MongoDB

MongoDB

Orientado a problemas de **búsqueda** mediante lectura indexada

- P.ej. Sitios de *eCommerce*

Foco: **Consistencia**

Arquitectura: Nodos *main-secondary*

Apache Cassandra

Almacenamiento rápido y disponibilidad inmediata

- P.ej. Spotify o Netflix

Foco **Escalabilidad** y **disponibilidad**

Arquitectura: *peer-to-peer*

Algunas ventajas de cassandra

Desarrollado directamente para ser un servidor **distribuido** y **descentralizado**

- Con soporte para distribución a lo largo de múltiples centros de datos

Lectura y escritura rápida sin afectar al servicio

Escalabilidad horizonal, permitiendo añadir nuevo hardware bajo demanda

- Alta disponibilidad sin punto único de fallo (SPOF, de *single point of failure*)
- Básicamente todo nodo del clúster conoce al resto

API simple para el acceso desde cualquier lenguaje de programación

Y algunas desventajas, claro

Las consultas sobre el sistema son *ad hoc*

- Se modela sobre las consultas a realizar, no sobre la estructura de los datos

No soporta agregaciones como parte del modelo subyacente

- No usan el álgebra relacional, donde las agregaciones son naturales
- Esto es, las operaciones (AVG, MAX, MIN, SUM) son **MUY** costosas

Rendimiento impredecible

- Ejecuta por debajo muchos procesos asíncronos y tareas de segundo plano
- Esto se traduce en un rendimiento impredecible

NO ES UN REEMPLAZO DIRECTO DE BASES DE DATOS RELACIONALES

Existen sitios donde se afirma esto de Apache Cassandra

- La principal causa: El lenguaje de consultas CQL

Sin embargo, no es cierto debido, principalmente, a tres factores:

1. No existe el concepto de **integridad referencia** (nada de *joins*)
2. El soporte a consultas agregadas es limitado
3. El soporte a transacciones es **muy** limitado

Al combinar con Apache Spark, se consiguen suplir los dos primeros puntos

¿Dónde usar Apache Cassandra?

Sistemas en el que las escrituras superan a las lecturas

- Almacenamiento de todas las peticiones realizadas a un sitio web

Cuando queremos, principalmente, añadir datos y no actualizar o borrar

Cuando nuestro acceso a datos se peude resumir a consultas por clave

- Esto permite que los datos se puedan particionar según la clave
- Y así, se pueden distribuir entre los nodos de un clúster

Cuando no necesitamos hacer *joins* o agregaciones

Ejemplos

Algunos ejemplos relacionados con aplicaciones web

- Actualización de datos provenientes de múltiples sensores
- Almacenamiento de transacciones comerciales
- Autenticación de usuarios
- Estado de las transacciones, los pedidos, etc.
- Interacciones con el sitio (clicks, votaciones, etc.)
- Monitorización de logs de acceso a sistemas
- Almacenamniento de perfiles de usuarios
- Tracking de actividad de usuarios en el sitio web

Sobre el consistencia, la disponibilidad y la tolerancia a fallos

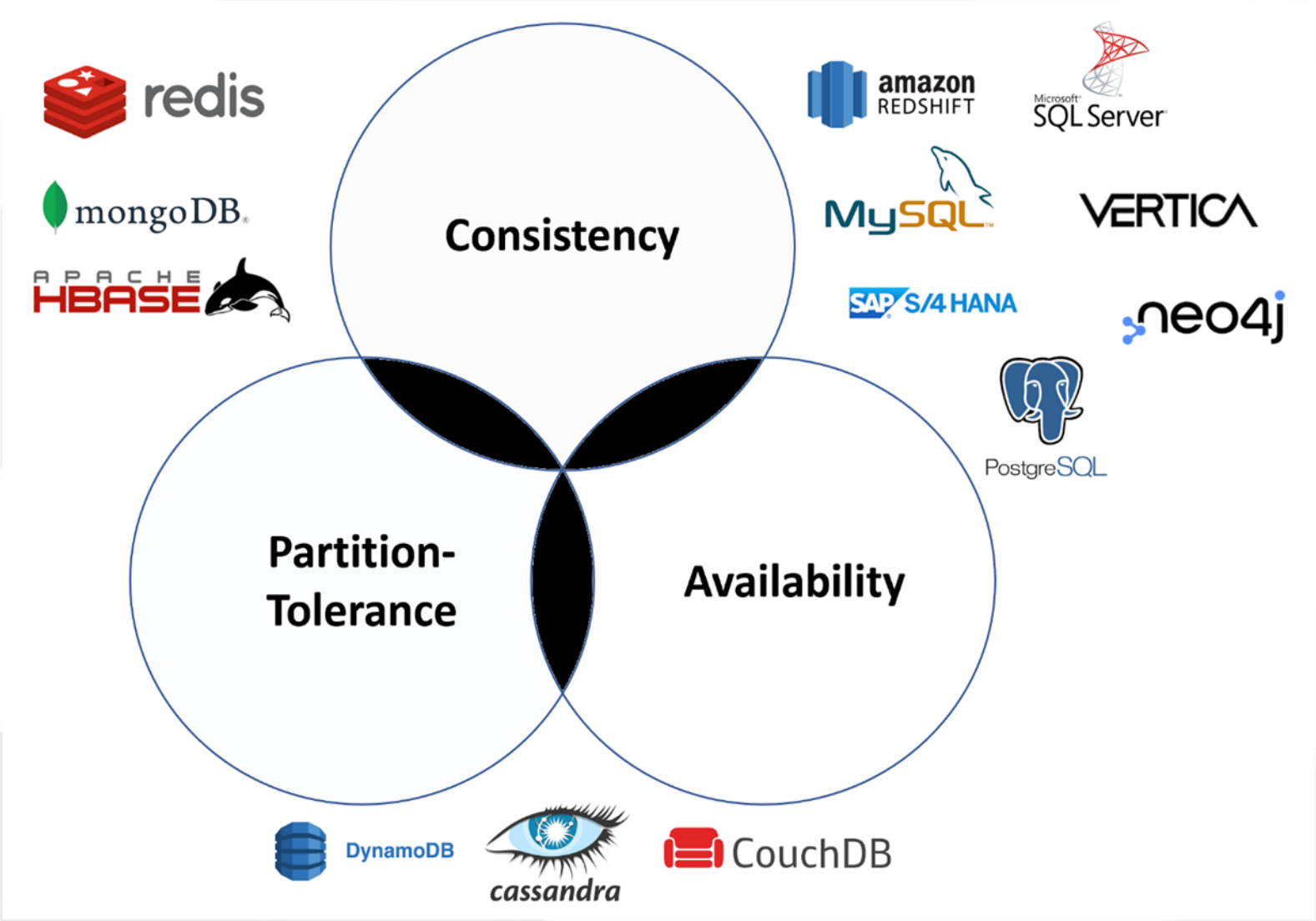
Teorema de Brewer

Establece que un sistema sólo puede garantizar a la vez dos requisitos entre:

- **Consistencia**: Toda lectura recibe la escritura más reciente o un error.
- **Disponibilidad**: Toda solicitud recibe una respuesta (no error)
- **Tolerancia al particionado**: Funcionamiento a pesar de pérdida de mensajes

Conocido también como **Teorema CAP** (***C****onsistency*, ***A****vailability*, ***P****artition tolerance*)

Teorema de Brewer



Teorema de Brewer y Cassandra

Desde un punto de vista práctico, la tolerancia a la partición es necesaria

- Los fallos en red son una constante en nuestra profesión
- Por ello las bases de datos suelen ser CP o AP

Cassandra es un sistema AP, es decir, prioriza disponibilidad sobre consistencia

- Esto es simplificar mucho, en realidad busca satisfacer los tres requisitos
- De hecho se puede configurar para que se comporte muy parecido a CP
    - Según la Apache Software Foundation, ofrece una **consistencia ajustable o eventual** del orden de los milisegundos.

Garantizando la disponibilidad de los datos

La disponibilidad se garantiza a través de las réplicas

- Cuando se escribe un dato, se escribe en varias réplicas
- Generalmente tres, aunque este parámetro es configurable
- Así nos aseguramos de que si cae un nodo los datos no se pierden

Las réplicas demoran la escritura

- Hay momentos en los que diferentes nodos tienen diferentes valores
- Casandra se describe como **eventualmente consistente** por esto mismo

Es decir, se garantiza la disponibilidad de los datos, pero no a su última versión

Tolerancia a fallos y alta disponibilidad

Ya hemos comentado que la arquitectura de comunicación es *peer_to_peer*

- Cuando algún nodo está caído, el resto de nodos lo detectan
- Esto produce una reconfiguración de la distribución de datos
- También de las *requests*, que se redirigen a los nodos disponibles

Esto es aplicable a la escalabilidad de un clúster

- Cuando se añade un nuevo nodo, también se redistribuyen los datos
- Estos nodos reciben rangos de tokens de otros nodos
    - Así las responsabilidades se reparten de forma equitativa
- **El rendimiento escala linealmente con el número de nodos**

Rendimiento en la escritura de datos

Cassandra está pensada, sobre todo, para la escritura de datos

- Su rendimiento no sólo se debe a que se lancen en paralelo a nivel de clúster
- A nivel de nodo:
  - Las escrituras se realizan en memoria (y luego se persisten en disco)
  - Todas las operaciones de escritura son secuenciales (como _appends)
  - Existe un proceso posterior de compactación que optimiza el almacenamiento

Además, no existe lectura antes de la escritura

- Otros sistemas lo suelen hacer, lo que ralentiza la escritura

Sistema distribuido y descentralizado

Distribuido y descentralizado

Prácticamente todas las bases de datos NoSQL son distribuidas

- Ahora bien, que sean distribuidas **Y** descentralizadas no es nada común

¿Qué es cada cosa?

- **Distribuido**: Los clústers pueden funcionar en diferentes máquinas
  - Eso sí, se muestran como un único sistema al exterior
- **Descentralizado**: Todo nodo es idéntico en responsabilidades
  - Dicho de otro modo, no hay nodos primarios o secundarios.

La comunicación entre nodos se realiza mediante *peer_to_peer* (protocolo *gossip*)

¿Cómo se distribuyen los datos? (I)

Id	Username	Password	Service
1	user1	pass1	**twitter**
2	user2	pass2	gmail
3	user3	pass3	**twitter**
4	user4	pass4	gmail
5	user5	pass5	**twitter**
6	user6	pass6	gmail

Tenemos un conjunto inicial de datos

- Supongamos que nuestras queries son sobre el campo *Service*
  - *¿Cuántos usuarios hay por servicio?*
- Esto requiere agrupar los datos por la columna *Service*
  - Es decir, **particionar** por *Service*
  - `PartitionKey = Service`

¿Cómo se distribuyen los datos? (y II)

Cassandra agrupa los datos de acuerdo a las claves de partición definidas

- Tras ello, los datos se distribuyen en los nodos del clúster
- Esta operación se realiza mediante el hash de cada clave de partición
    - Se denominan **tokens**

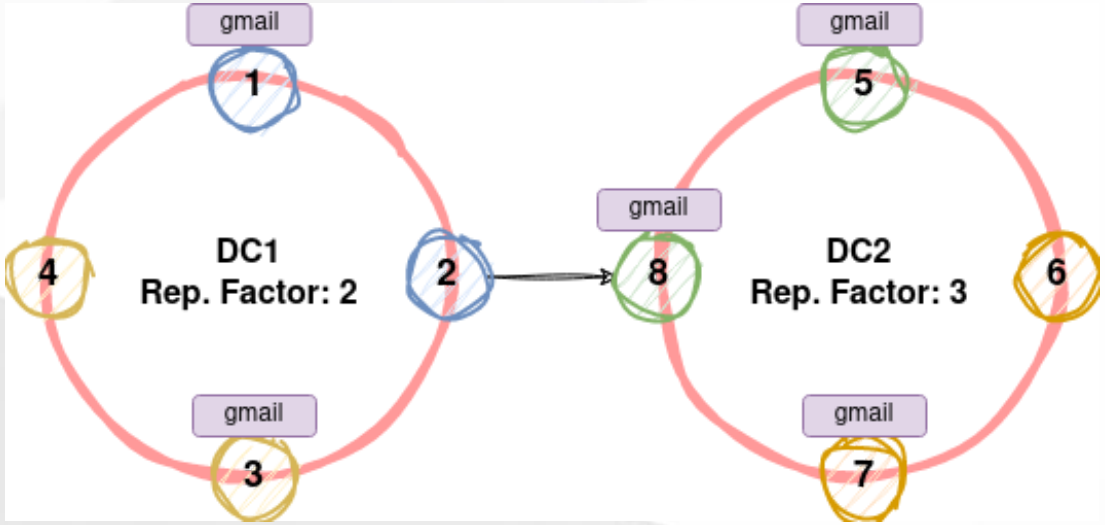
Cada nodo tiene una lista predefinida de intervalos de *tokens* soportados

- Los datos se dirigen al nodo adecuado en función de:
    i. El valor hash de la clave
    ii. La preasignación del clúster (los intervalos de *tokens* predefinidos)

¿Cómo se replican los datos?

Tras la asignación inicial de datos, Cassandra se encarga de replicarlos

- Esta se realiza de acuerdo al factor de réplicación (*replication factor*)
- Indica el número de nodos que mantendrán réplicas para cada partición



Suponiendo que los nodos están distribuidos en:

- Rack 1 (1, 2), Rack 2 (3, 4), Rack 3 (5, 6), Rack 4 (7, 8)

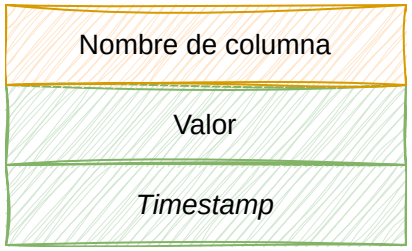
La replicación tratará de hacerse abarcando todos los racks

Algunos conceptos de Cassandra

Columna (*column*)

La unidad básica de almacenamiento

- Es una tupla que contiene un nombre, un valor y un *timestamp*
- Se pueden agrupar en súpercolumnas, familias de columnas y *keyspaces*
- Son análogas a las columnas de una tabla en el modelo relacional

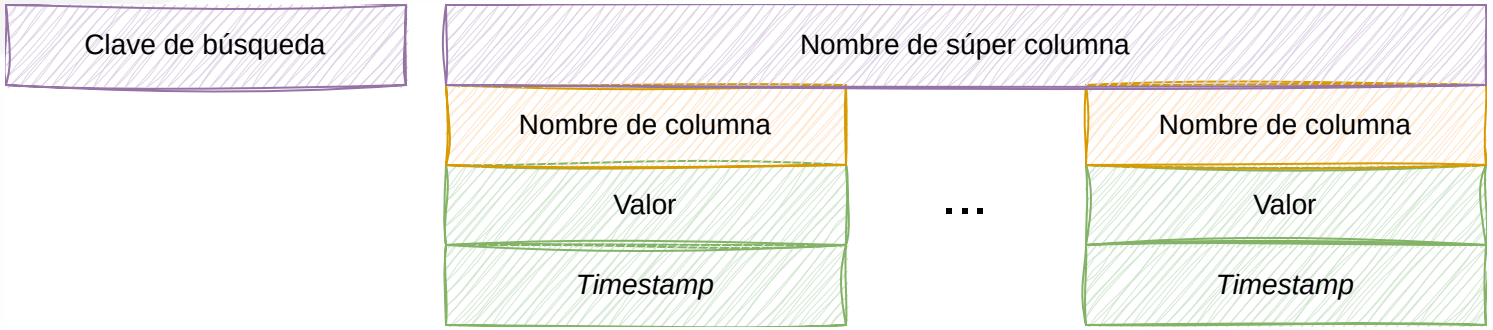


El *timestamp* indica el momento de la última actualización de la columna

Súpercolumna (*super column*)

Array ordenado de columnas

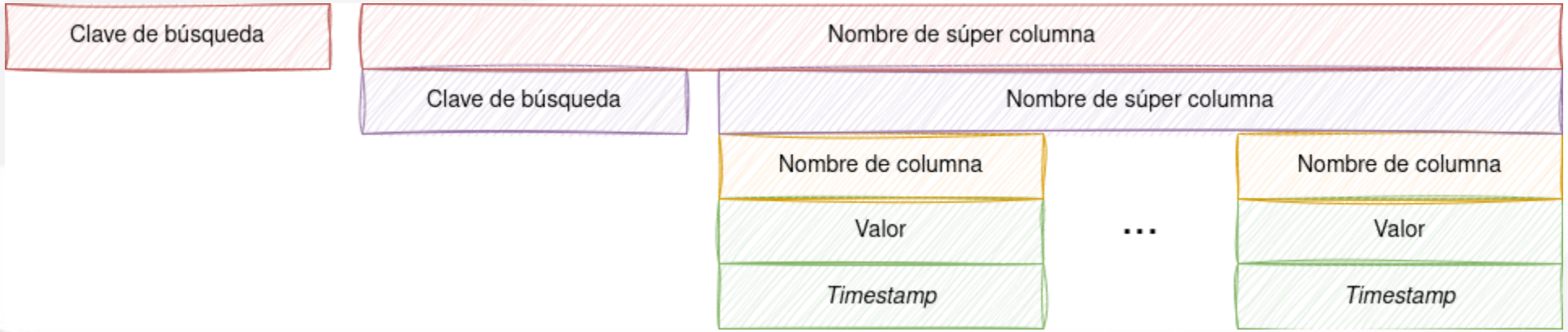
- Se compone de una clave (*row key*), un nombre y su conjunto de columnas



Familia de columnas (*column family*)

Agrupación de columnas o súper-columnas que comparten una clave común

- Conjunto de pares clave-valor donde el valor es una lista de nombres de columnas



Aunque gestione columnas y familias de columnas **no es una bbdd columnar**

Clúster o anillo

Conjunto de nodos que componen una instancia de Cassandra

- Anillo hace referencia a la topología que se formaba en el pasado
- Ahora, con los nuevos nodos, más "enredada"
    - Prácticamente todos los nodos hablan con todos

Keyspaces (I)

Son contenedores que almacenan los datos que usa la aplicación

- Esquema de alto nivel que **contiene familias de columnas**
  - Debe haber al menos una familia de columnas por espacio de claves

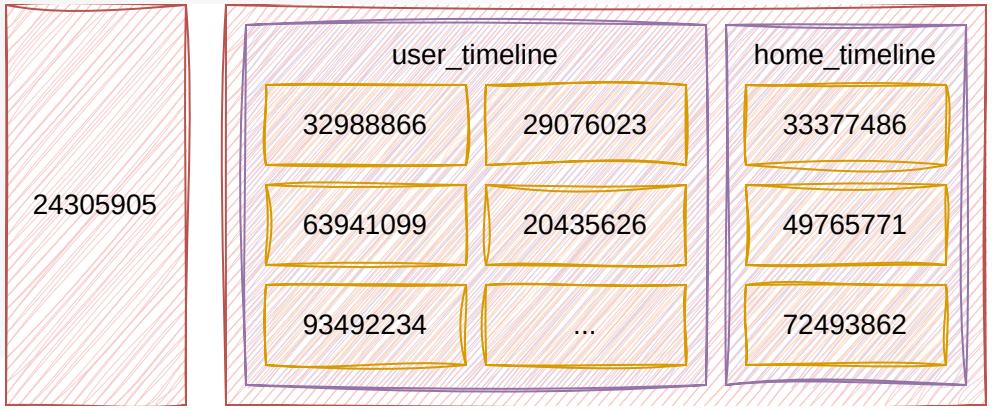
Análogos a una base de datos en el modelo relacional (pero claro, sin relaciones)

Requieren una configuración de atributos relacionados con la consistencia:

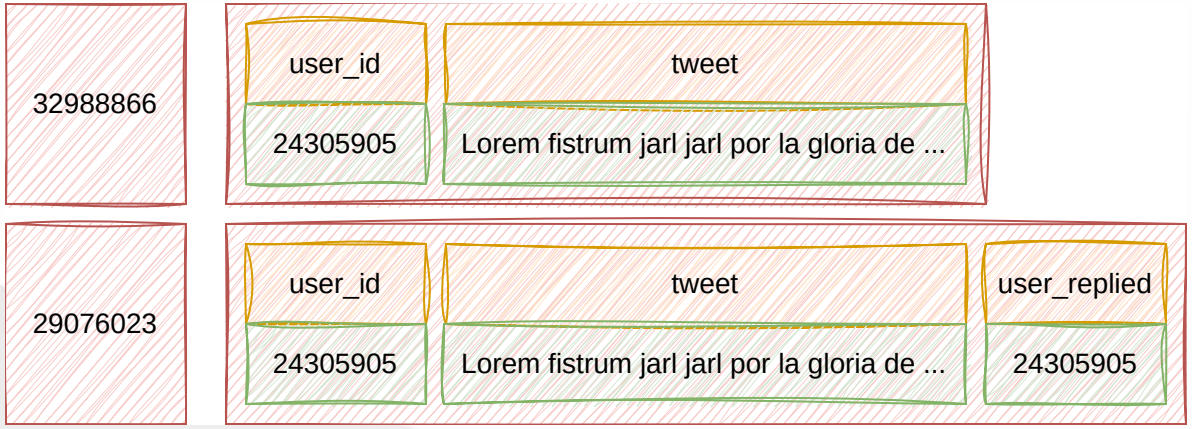
- **Factor de replicación**: Cuánto mejorar la coherencia a cosa del rendimiento
- **Estrategia de ubicación de réplica**: Cómo se colocan las réplicas (`SimpleCategory`, `OldNetwork` `TopologyStrategy` y `NetworkTopologyStrategy`)

Keyspaces (y II)

Familia de supercolumnas para `user_status`



Familia de columnas para `user_timeline`



Cassandra Query Language (CQL)

Cassandra Query Language (CQL)

Es el lenguaje de Cassandra para la definición y manipulación de datos

- Es un lenguaje de consulta de alto nivel, muy similar a SQL
  - Los datos se almacenan en tablas que contienen filas de valores
  - Por ello la terminología (tabla, fila y columna) es la misma en CQL

```
CREATE TABLE Accounts (
  Id int PRIMARY KEY,
  Username text,
  Password text
  Service text
  );

INSERT INTO Accounts (Id, Username, Password, Service)
  VALUES (1, 'user1', 'pass1', 'twitter'), (2, 'user2', 'pass2', 'gmail');

SELECT * FROM accounts WHERE Service = 'twitter';
```

Algunas características

Algunas de las características más notables de CQL son:

- Tipos de datos y operadores aritméticos
- DDL y DML
- Índices secundarios
- Vistas materializadas
- Funciones y triggers

Pero cuidado, que la sintáxis similar a SQL no nos lleve a engaño:

- No es una base de datos relacional
- Las lecturas y escrituras de datos son muy diferentes, y esto afecta a la forma de escribir las consultas
    - No nos preocupemos que lo veremos más adelante

Gracias