EAE Business School

Cassandra

Máster en Business Intelligence e Innovación Tecnológica

Índice de contenidos

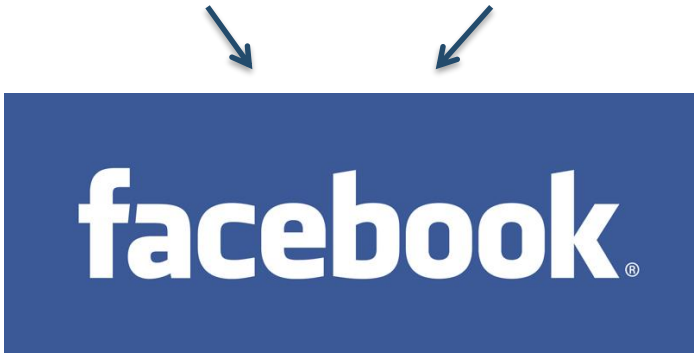
Introducción

- Cassandra es una base de datos distribuida, tolerante a fallos, escalable y orientada a columnas.
- Es una solución post relacional.
- Sirve acceso a datos tanto para motores online / transaccionales como para motores de lectura intensiva para BI.

Bigtable

Dynamo

Toma:
Modelo de datos y de almacenamiento en disco



Toma:
Arquitectura de clúster distribuido

DATASTAX

Cassandra

Who *else* Uses Cassandra?

Atributos clave

- Escalabilidad Big Data.
- Rendimiento rápido y lineal.
- Replicación: no existe un único punto de fallo.
- Distribución de datos: Enterprise / multi-data center / Cloud.
- Capacidad de lectura y escritura en cualquier nodo.
- Flexibilidad de esquemas.
- Consistencia de datos parametrizable.
- Compresión de datos.
- Lenguaje familiar a SQL: CQL.
- Instalación sencilla.
- No necesita hardware especial.
- No necesita una capa de caché adicional.

El teorema CAP

- Las bases de datos derivadas de Amazon Dynamo incluyen Cassandra, Voldemort, CouchDB y Riak
- Centradas más en disponibilidad y tolerancia a fallos
- Permiten Consistencia Eventual
  - Donde “eventual” significa milisegundos
  - Por tanto Cassandra es AP:

> “To primarily support Availability and Partition Tolerance, your system may return inaccurate data, but the system will always be available, even in the face of network partitioning”

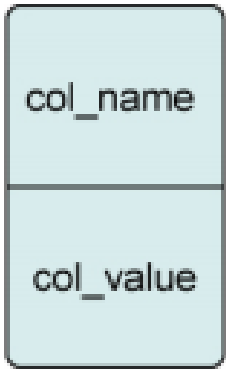
Modelo de datos

- Diseñado para datos distribuidos de modo escalable, sacrifica ACID por ventajas en rendimiento, disponibilidad y gestión operacional
- Los modelos que se crean son desnormalizados:
  - Se suele crear una column family por cada consulta (query) a realizar
  - Varias filas en un column family suelen dar respuesta a una consulta
- Los conceptos básicos son:
  - Clúster: son las máquinas que componen una instancia de Cassandra (pueden contener varios Keyspaces)
  - Keyspace: espacio de nombres para un conjunto de ColumFamily, asociado a una aplicación (suele vincularse con una BBDD en el modelo relacional)
  - ColumFamily: contienen varias columnas (suelen vincularse con una tabla en el modelo relacional)
    - SuperColumn: columnas que ellas mismas tienen sub-columnas
    - Column: compuestas de un nombre, valor y timestamp

Column

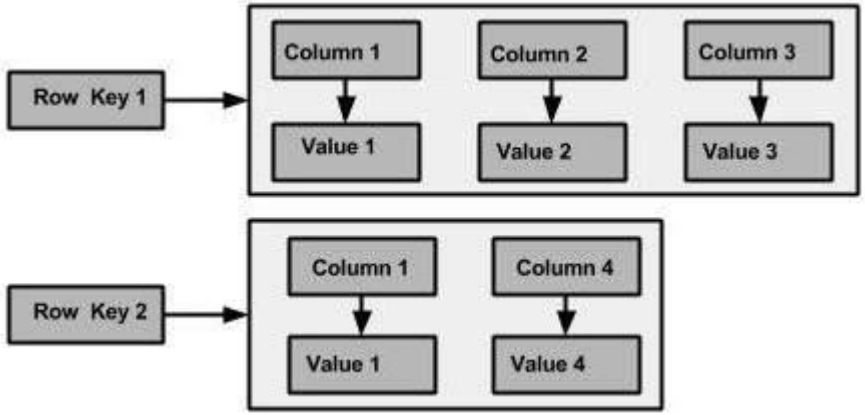
- Una columna es un par nombre-valor que también contiene un timestamp
  - Los nombres y columnas son arrays de bytes
  - El timestamp registra la última vez que una columna es accedida
  - Unidad atómica
    - name:value:timestamp
    - Email:joan.fustero@campus.eae.es:123456789
- Ejemplo en JSON:

```
{
    "name": "Email",
    "value": "joan.fustero@campus.eae.es",
    "timestamp”: 123456789
}
```



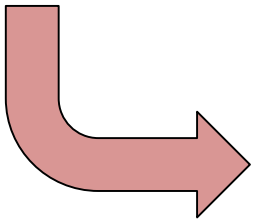
Column family

- Es un contenedor de columnas (Análogo al concepto de tabla en RDBMS)
- Contiene una lista ordenada de columnas
  - Cuando se crea de manera configurativa una familia de columnas se indica cómo se ordenarán las columnas de cada fila (cogiendo el nombre) •
- Cada familia de columnas se guarda en un fichero, y el fichero está ordenado por clave de fila
- Una familia de columnas tiene un … conjunto de filas con un conjunto de … columnas similar pero no idéntico
- Pueden ser de tipo SUPER o STANDARD



Column family

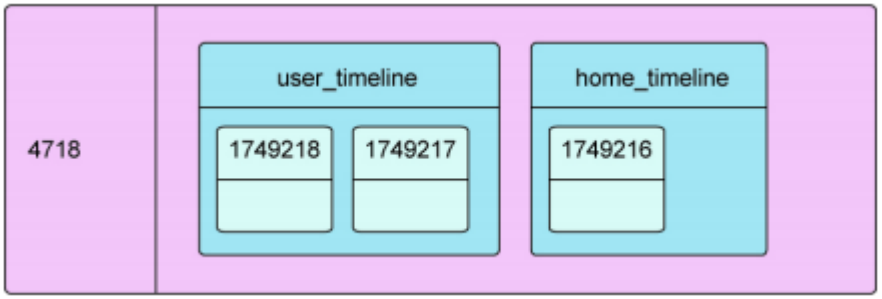
id	name	state	birth_date
7b976c48...	Bill Watterson	DC	1953
7c8f33e2...	Howard Tayler	UT	1968
7d2a3630...	Randall Monroe	PA	
7da30d76...	Dave Kellett	CA	



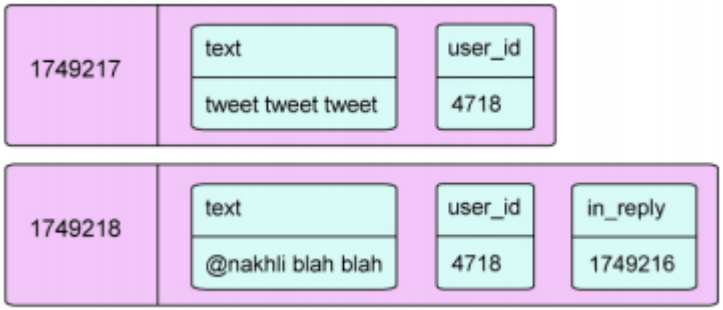
7b976c48...	name: Bill Watterson	state: DC	birth_date: 1953
7c8f33e2...	name: Howard Tayler	state: UT	birth_date: 1968
7d2a3630...	name: Randall Monroe	state: PA	
7da30d76...	name: Dave Kellett	state: CA	

Keyspaces

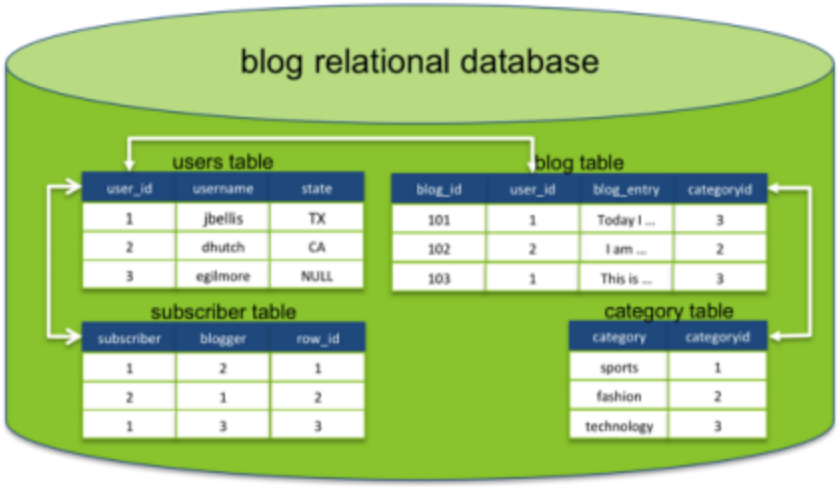
- Un espacio de claves o KeySpace es un esquema de alto nivel que contiene familias de columnas.
- Supercolumn Family “Estado de Usuario”

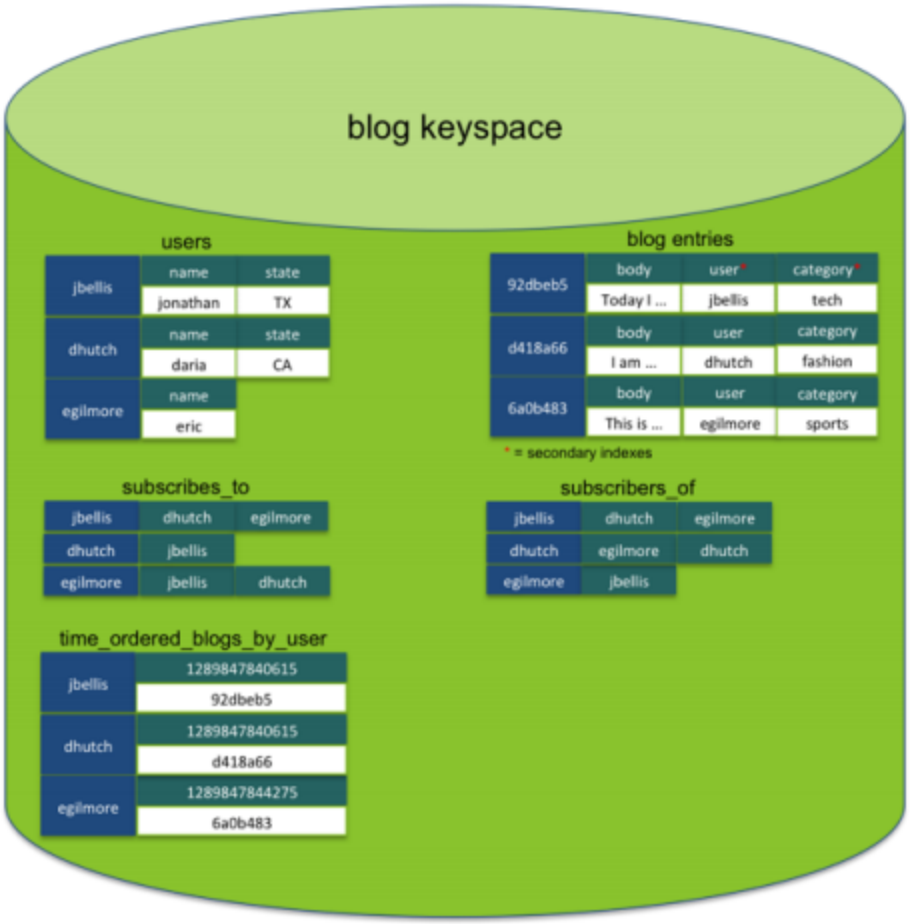


- Column Family “Entradas en Twitter: tweets”:



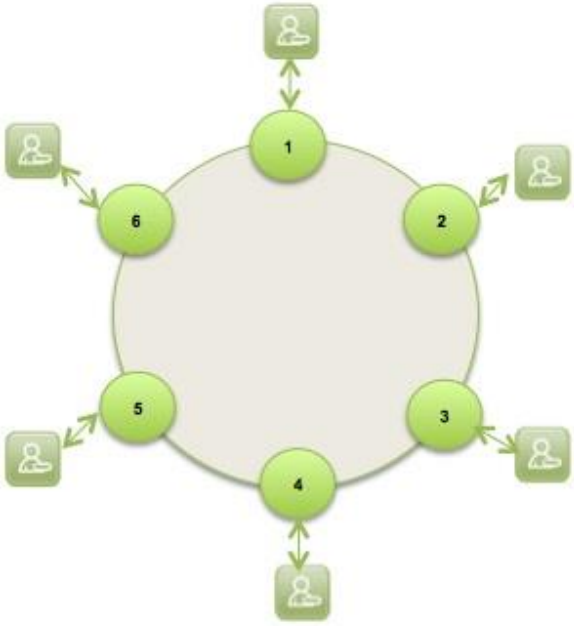
Keyspaces vs RDBS





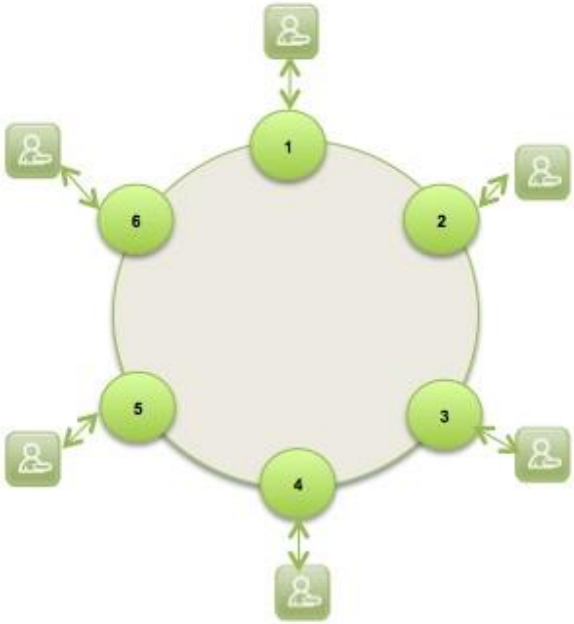
Resumen

- Esta diseñada entendiendo que los fallos de Sistema pueden ocurrir.
- Define un Sistema distribuido punto a punto:
  - La información se distribuye en nodos, servidores individuales equivalentes entre ellos.
  - Los nodos se agrupan en datacenters, en función de la carga, de la replicación, etc.
  - Los clusters agrupan uno o más datacenters, en una o varias localizaciones físicas.
- La información se replica para asegurar la tolerancia a fallos.

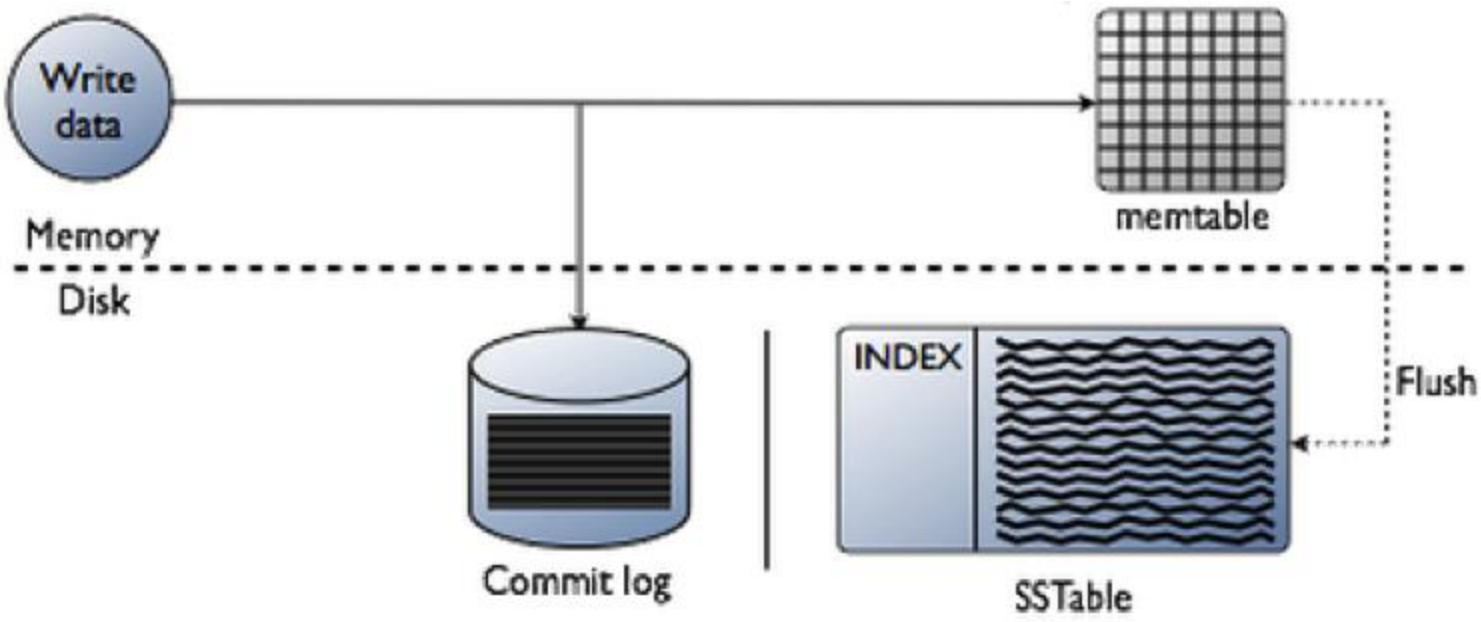


Resumen

- Diseño Read/Write-anywhere.
- Cada nodo se comunica con el otro a través del protocolo *Gossip*, quién intercambia información a través del cluster cada Segundo.
- Los commits en cada nodo se capturan para registrar las actividades de escritura: la durabilidad de los datos está asegurada.
- La información se escribe en una estructura de datos en memoria (*memtable*) y posteriormente a disco cuando se llena (*sstable*).

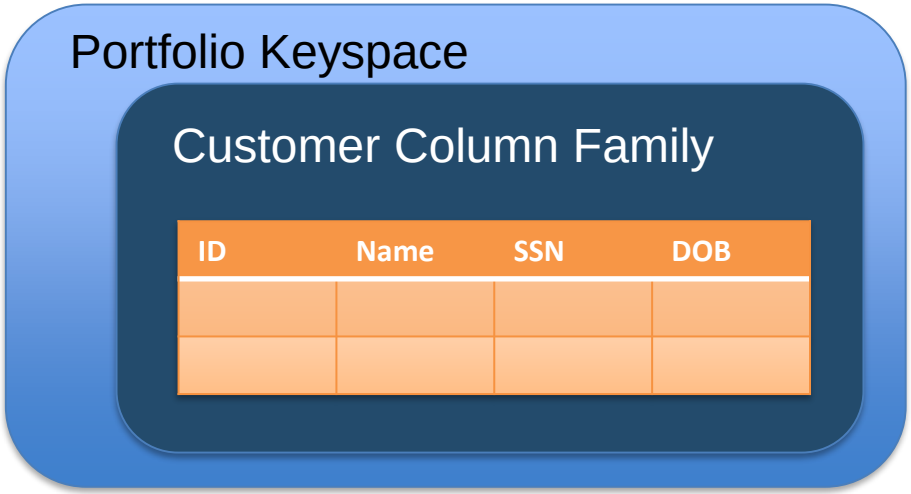


Resumen


Write data
memtable
Memory
Disk
Flush
INDEX
Commit log
SSTable

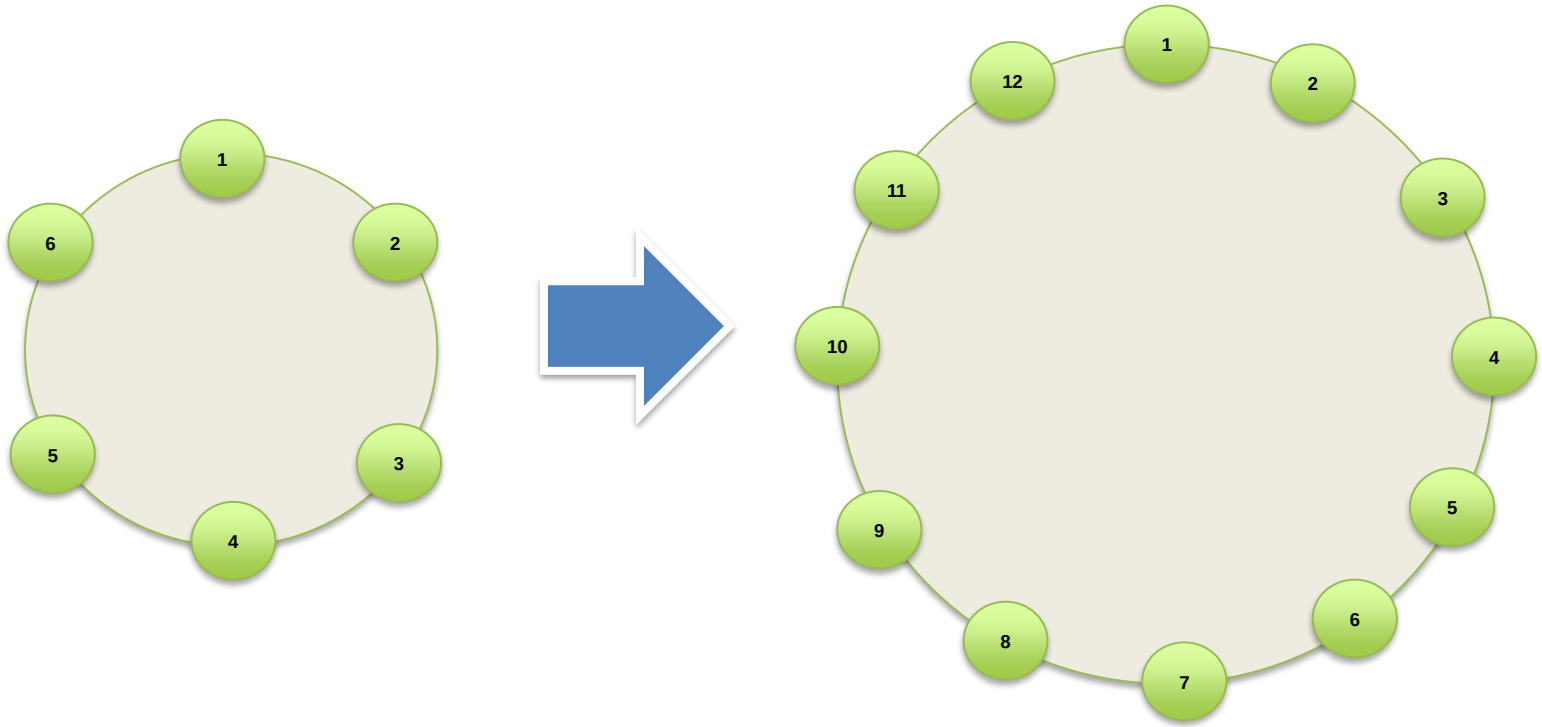
Resumen

- El esquema utilizado en Cassandra nace a partir de Google BigTable: orientado a filas, las columnas pueden almacenar información estructurada, semi-estructurada y no estructurada.
- El keyspace es similar a una base de datos en el mundo RDBMS.
- Una familia de columnas es similar a una table relacional, pero más flexible y dinámica.
- Cada fila en una familia de columnas se indexa por su clave, las columnas también se pueden indexer.



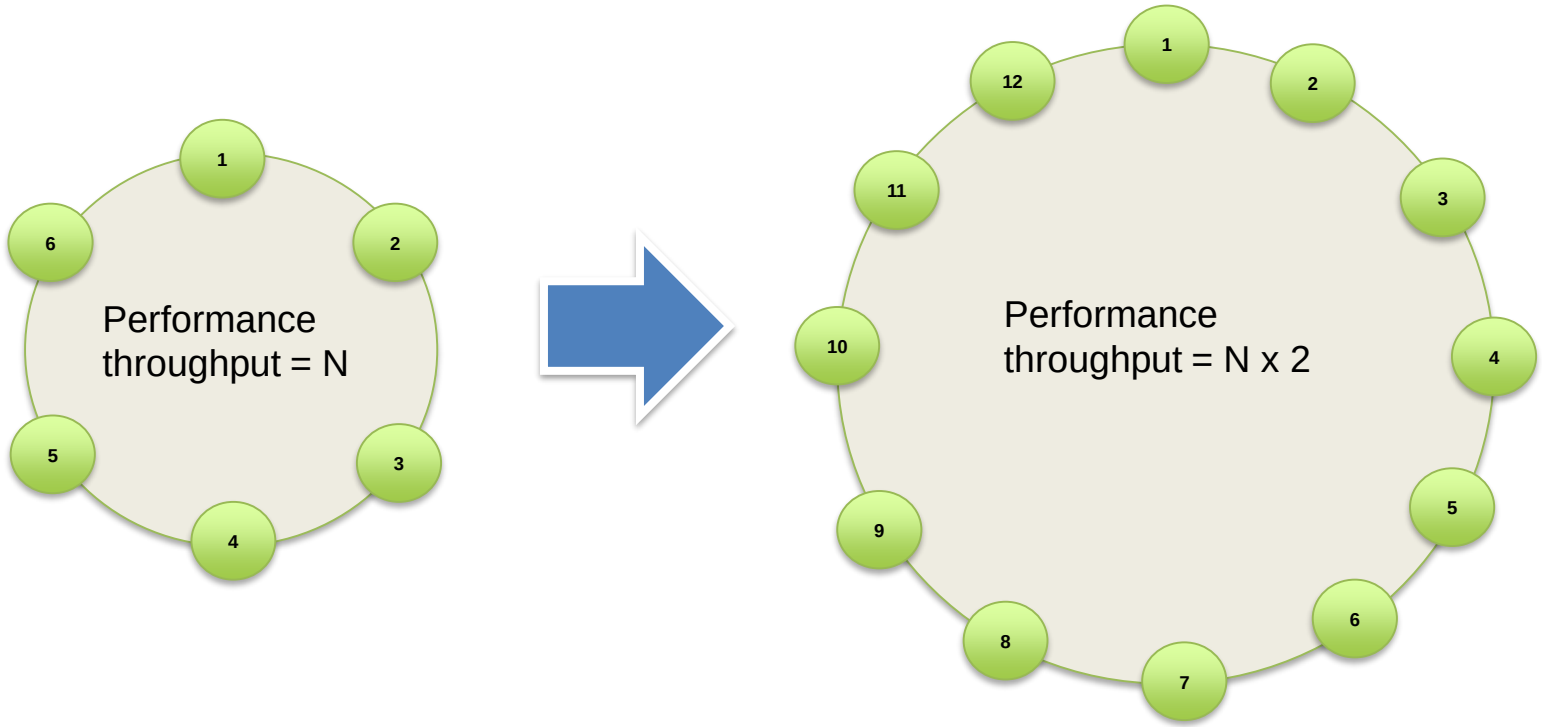
Elasticidad transparente

- Es la posibilidad de añadir y eliminar nodos del motor de Cassandra en un modo online, sin retrasos ni impacto para el usuario.



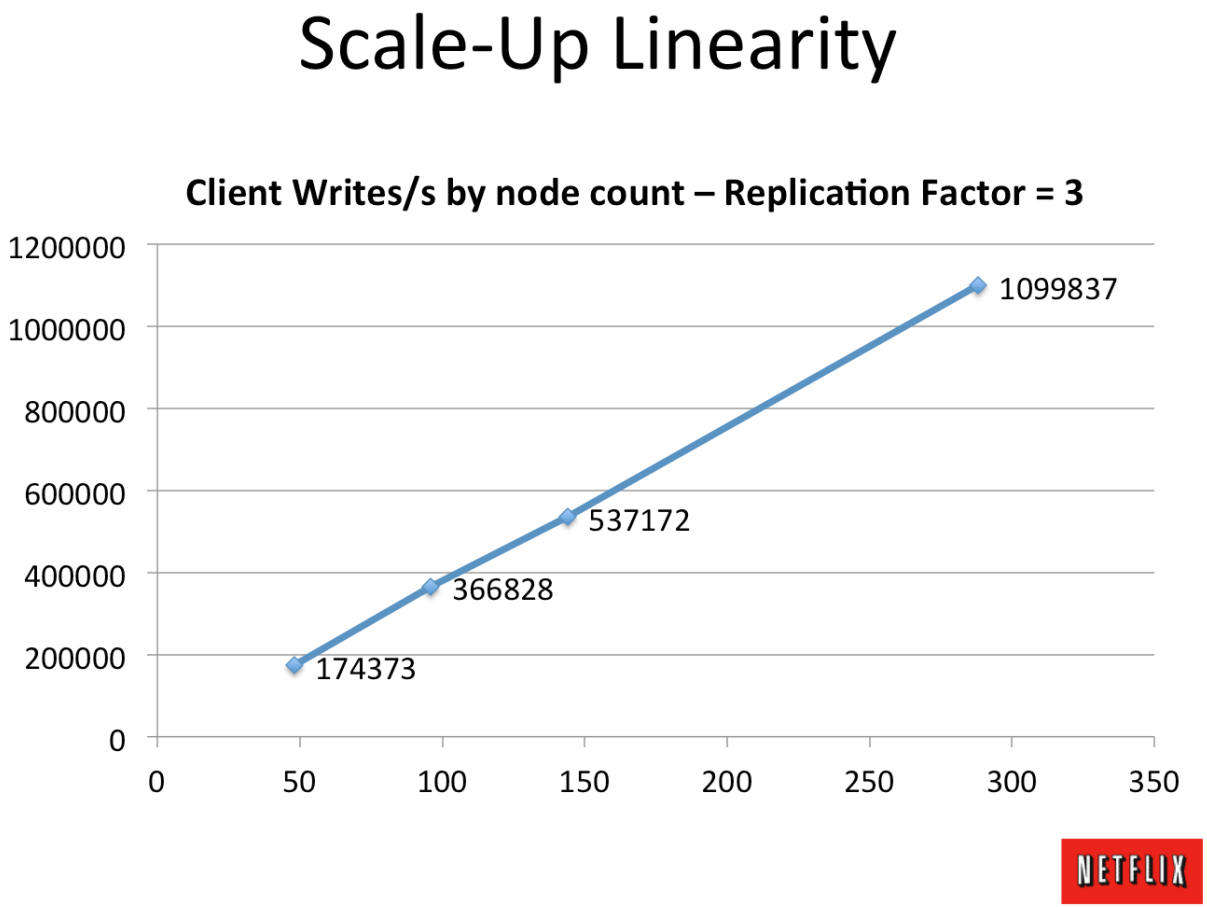
Escalabilidad transparente

- La adición de nodos en Cassandra incrementa el rendimiento de forma lineal y la habilidad de manejar Terabytes y Petabytes de información.



Escalabilidad transparente

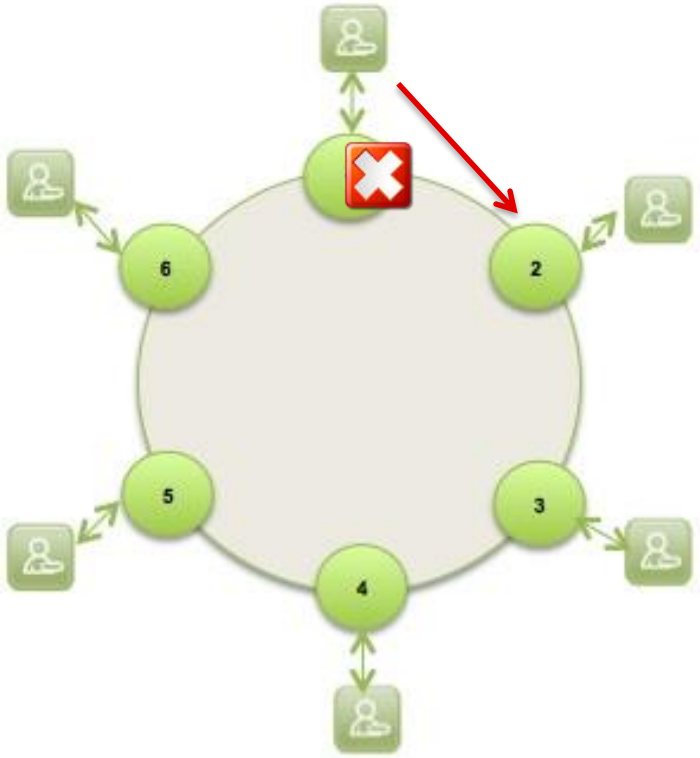
> Millón escrituras / s



http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html

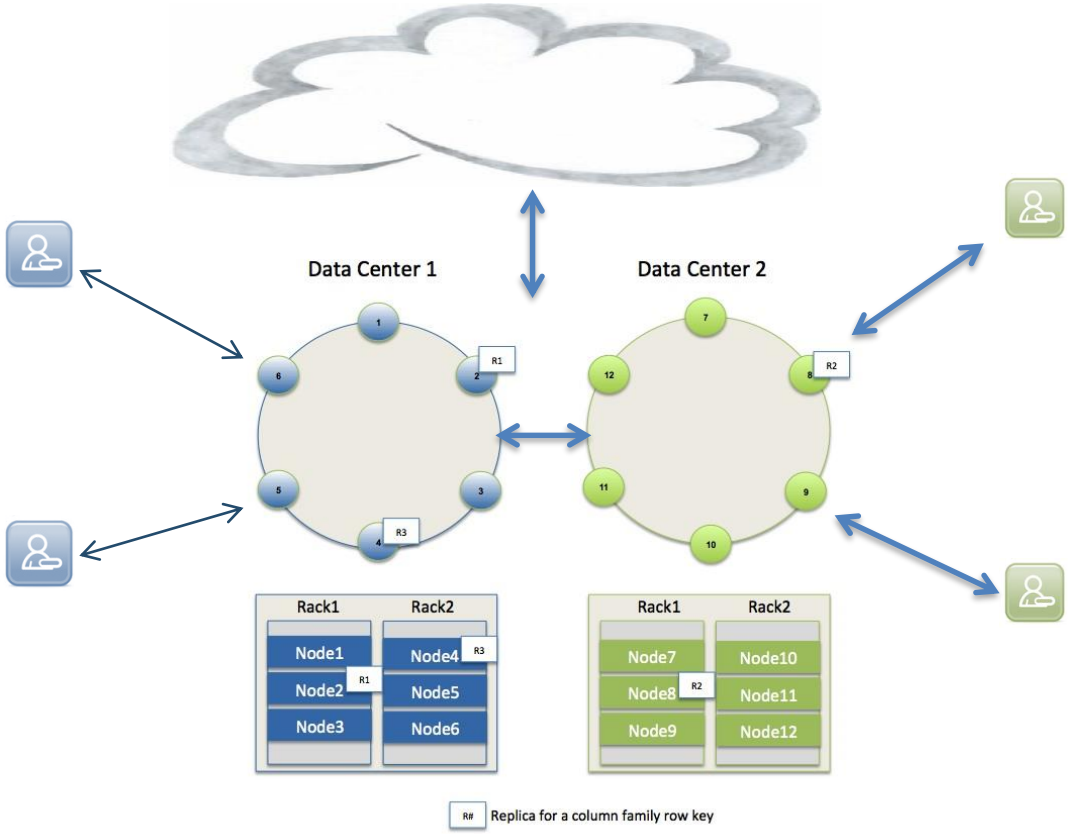
Alta disponibilidad

La arquitectura punto a punto de Cassandra proporciona que no exista un punto de error (SPoF, *Single Point of Failure*).



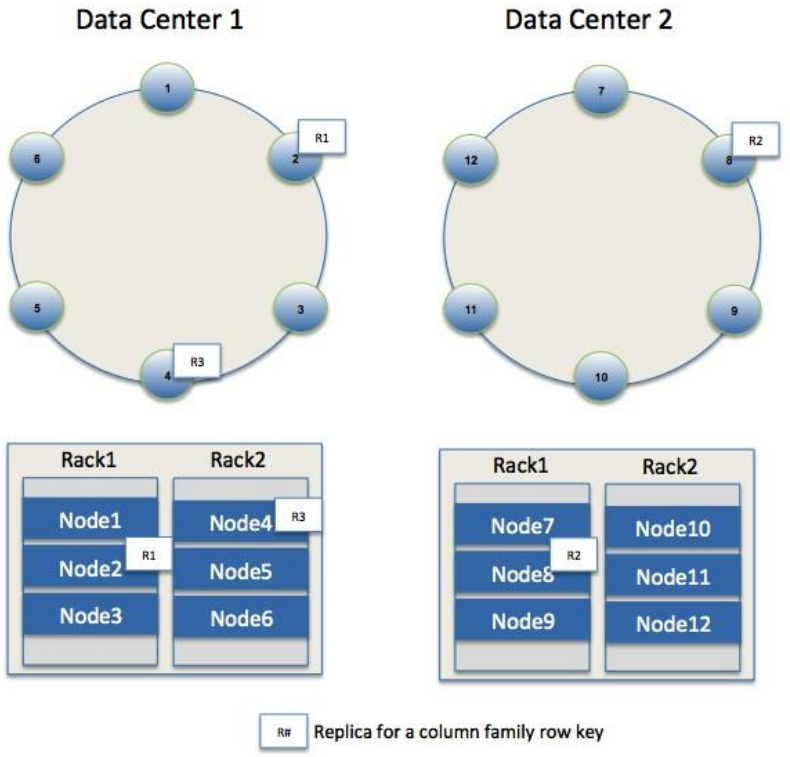
Multi-geografía

- Cassandra también permite expander una única base de datos a N datacenters (clusters) dispuestos de forma dispersa geográficamente y con replicación de datos.
- También permite una implementación en una cloud híbrida.



Redundancia de datos

- Cassandra permite una redundancia de datos customizable de forma que la información esté totalmente protegida.
- También permite replicación multi-rack y multi-datacenter, ganando abstracción de cara a errores en máquinas puntuales.



Introducción

Como se ha comentado, una de las características de Cassandra es la de la replicación de datos: la información se distribuye entre varios nodos de forma que el Sistema se vuelva más robusto ante los posibles fallos.

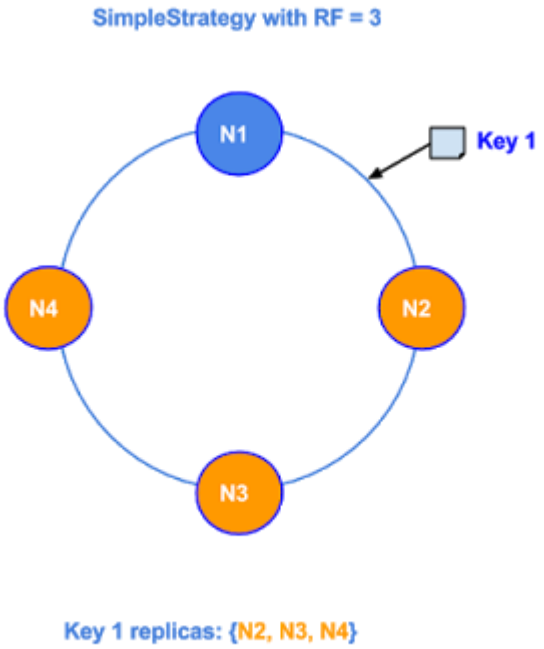
Para ello, Cassandra utiliza dos estrategias:

- **SimpleStrategy**
- **NetworkTopologyStrategy**

El factor de replicación (RF) identifica cuántas replicas de cada element existen en el Sistema.

SimpleStrategy

- Concebida para en modo de un único rack y datacenter.
- Los nodos siguientes en el anillo son seleccionados para guardar las replicas.

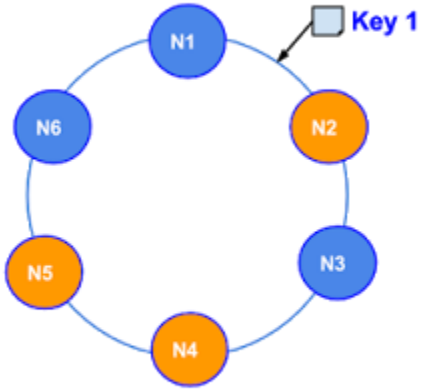


NetworkTopologyStrategy

- Nodos de distintos racks y datacenters son elegidos para contener las réplicas.
- El usuario define el factor de replicación para cada datacenter.

NetworkTopologyStrategy with Replication factor
{ DC1: 2, DC2: 2 }

Node	DC	RACK
N1	DC1	RACK2
N3	DC1	RACK1
N6	DC1	RACK1
N2	DC2	RACK1
N4	DC2	RACK1
N5	DC2	RACK2

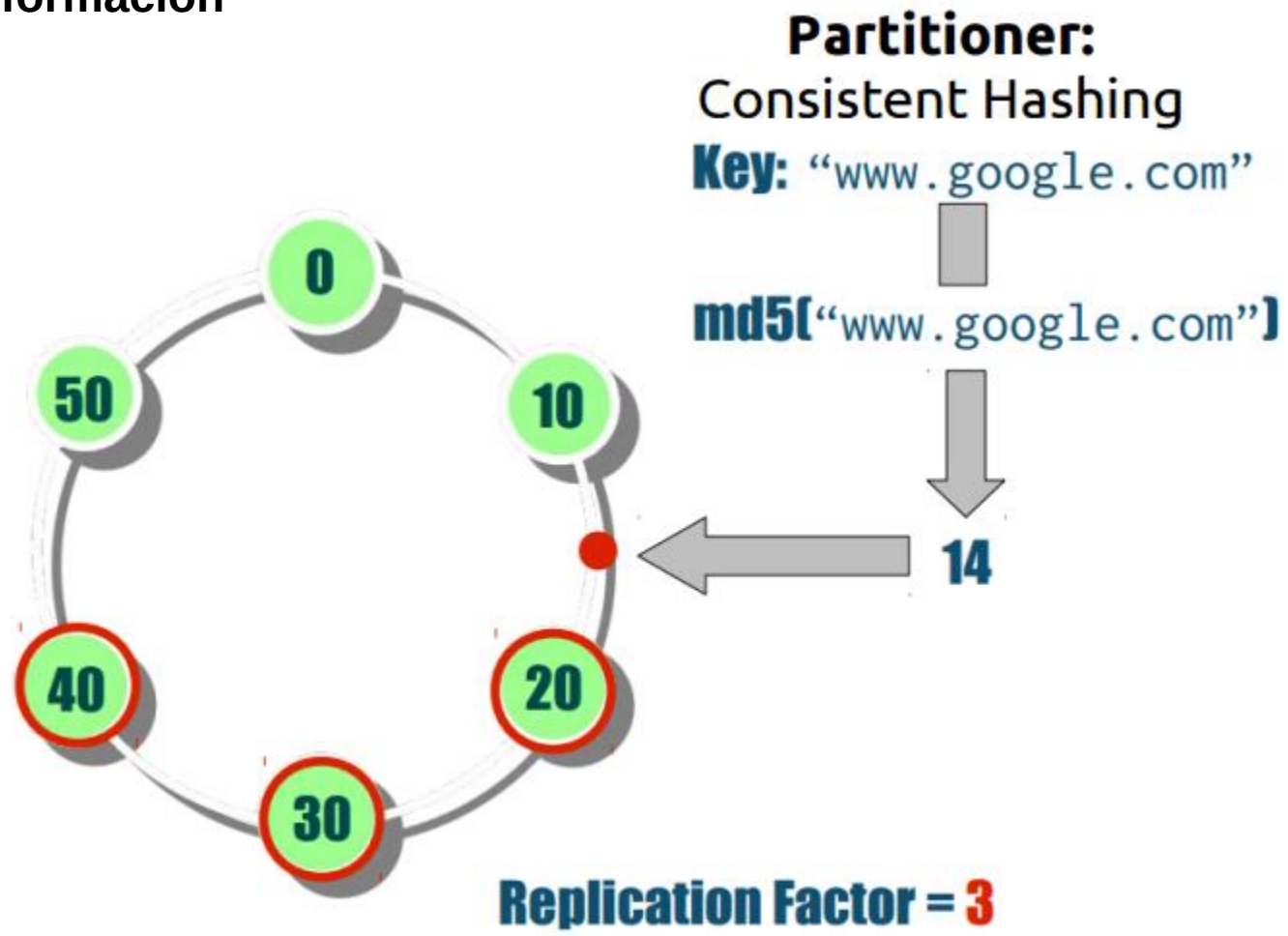


Key 1 Replicas DC1 : {N3, N1} DC2: {N2, N5}

Búsqueda de información

- Cada nodo tiene un token (identificador) que representa el identificador más alto que contiene.
- Gossip proporciona a cada nodo la capacidad de saber qué contienen los otros nodos.
- Los clientes acceden a cualquier nodo y buscan por cierto identificador. El nodo al que acceden actuará como coordinador.
- El coordinador conocerá quien almacena esa clave y dirigirá la petición hacia ese nodo, quien devolverá la información.
- En caso de error o indisponibilidad, se buscarán las fuentes registradas como contenedores de réplicas.

Búsqueda de información



Nodetool

- Nodetool es una herramienta distribuida con Cassandra que muestra la información de los nodos de un datacenter de Cassandra.

```
user@ubuntu:~$ sudo nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens       Owns (effective)  Host ID
                Rack
UN  127.0.0.1  126.91 KB  256          100,0%            451f6bbc-68fe-42fc-8ee2
-1a9ad0039061   rack1
```

Introducción

- CQL es un lenguaje propio de Cassandra con una sintaxis similar a la de SQL.

Podemos lanzar su consola desde el comando:

```
user@ubuntu:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.2.8 | CQL spec 3.3.1 | Native protocol v4]
Use HELP for help.
cqlsh>
```

Describe

- describe cluster: muestra información del cluster

```
cqlsh> describe cluster

Cluster: Test Cluster
Partitioner: Murmur3Partitioner
```

- describe keyspaces: muestra los keyspaces del cluster

```
cqlsh> describe keyspaces

system_traces  system_auth  twissandra  system  system_distributed
```

Describe

- describe tables: muestra las tablas de cada keyspace

```
cqlsh> describe tables

Keyspace system_traces
----------------------
events  sessions

Keyspace system_auth
--------------------
resource_role_permissons_index  role_permissions  role_members  roles

Keyspace twissandra
-------------------
users  timeline  followers  tweets  userline  friends
```

Describe

- describe table: muestra la información de una tabla

```
cqlsh> describe table twissandra.tweets

CREATE TABLE twissandra.tweets (
    tweet_id uuid PRIMARY KEY,
    body text,
    username text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = '{"keys":"ALL", "rows_per_partition":"NONE"}'
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}
    AND compression = {'sstable_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99.0PERCENTILE';
```

Create keyspace

```
cqlsh:ntskeyspace> CREATE KEYSPACE test WITH REPLICATION = { 'class' : 'SimpleSt
rategy', 'replication_factor' : 2 };
cqlsh:ntskeyspace> use test
                ... ;
cqlsh:test>
```

```
cqlsh>
cqlsh> CREATE KEYSPACE NTSkeyspace WITH REPLICATION = { 'class' : 'NetworkTopolo
gyStrategy', 'datacenter1' : 1 };
cqlsh> use NTSkeyspace
   ... ;
cqlsh:ntskeyspace>
```

- Definición de la estrategia de replicación:

{ 'class' : 'SimpleStrategy', 'replication_factor' : <integer> };

{ 'class' : 'NetworkTopologyStrategy'[, '<data center>' : <integer>, '<data center>' : <integer>] . . . };

Trabajando con datos

- Como se ha comentado, CQL tiene una sintaxis muy similar a SQL, por lo que para la creación de tablas el comando también lo será:

```
cqlsh:test> create table emp(
        ... empID int,
        ... deptID int,
        ... first_name varchar,
        ... last_name varchar,
        ... PRIMARY KEY (empID, deptID));
cqlsh:test>
```

- Con el comando anterior, ya tendríamos la tabla emp creada.

Trabajando con datos

- También podemos insertar datos de una forma muy similar a SQL:

```
cqlsh:test> insert into emp (empID, deptID, first_name, last_name)
        ... values (104, 15, 'jane', 'smith');
cqlsh:test>
```

- Y hacer lo propio con las sentencias SELECT:

```
cqlsh:test> select * from emp;

 empid | deptid | first_name | last_name
-------+--------+------------+-----------
   104 |     15 |       jane |     smith

(1 rows)
cqlsh:test>
```

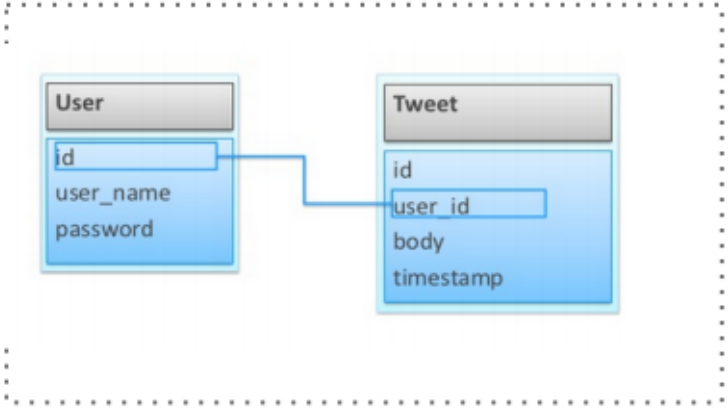
Ventajas

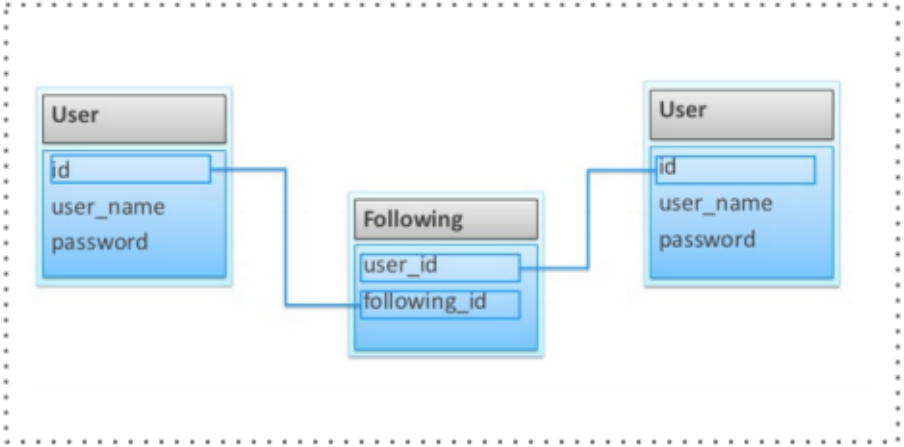
- Escalabilidad horizontal (añade nuevo hardware cuando sea preciso)
- Rápidas respuestas aunque la demanda crezca
- Elevadas velocidades de escritura para gestionar volúmenes de datos incrementales
- Almacenamiento distribuido
- Capacidad de cambiar la estructura de datos cuando los usuarios demandan más funcionalidad
- Una API sencilla y limpia para tu lenguaje de programación favorito
- Detección automática de fallos
- No hay un punto de fallo único (cada nodo conoce de los otros)
- Descentralizada
- Tolerante a fallos
- Permite el uso de Hadoop para implementar Map Reduce

Desventajas

- Hay algunas desventajas que un sistema de almacenamiento tan escalable ofrece en contrapartida:
  - No hay joins (a cambio de más velocidad)
  - No permite ordenar resultados en tiempo de consulta
  - No tiene SQL (CQL desde la versión 0.8)

Twissandra - RDBMS Version



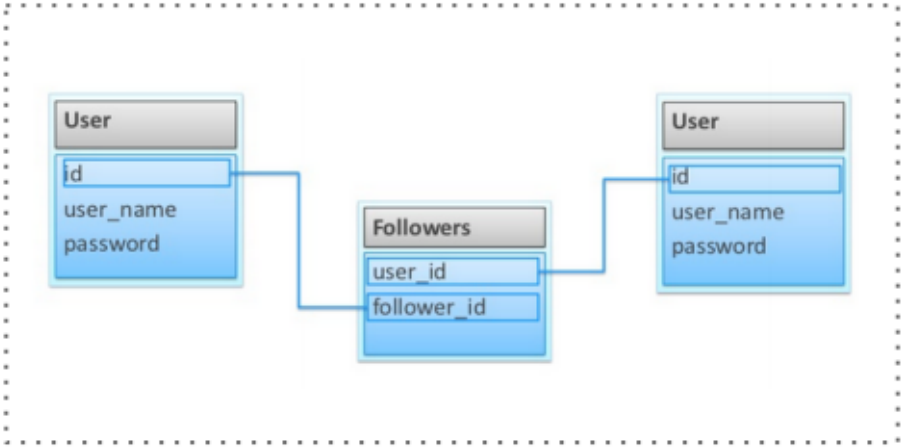


Entities

- USER, TWEET
- FOLLOWER, FOLLOWING
- FRIENDS

Relationships:

- USER has many TWEETs.
- USER is a FOLLOWER of many USERs.
- Many USERs are FOLLOWING USER.

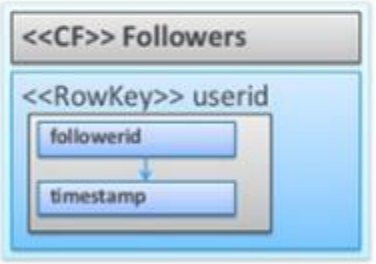


Twissandra - Cassandra Version

Tip: Model tables to mirror queries.

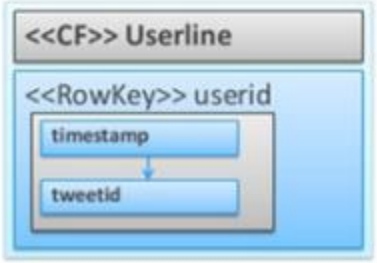
TABLES or CFs

- TWEET
- USER
- FOLLOWERS, FOLLOWING
- USERLINE, TIMELINE



Notes:

- Extra tables mirror queries.
- Denormalized tables are "pre-formed"for faster performance.



```
cqlsh:twissandra> select * from tweets;

 tweet_id                             | body           | username
--------------------------------------+----------------+----------
 35976bee-c589-41ec-a8f3-9c8d017853db |         hola!! |    Josep
 c1e3c2f3-56d8-4cf8-9c00-71d3a84492fd | hello friends! |     Joan
 cd31babf-3b7a-473f-beb6-904ca21589a8 |   hola @josep! |     Joan
```

```
cqlsh:twissandra> describe table tweets;

CREATE TABLE twissandra.tweets (
    tweet_id uuid PRIMARY KEY,
    body text,
    username text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = '{"keys":"ALL", "rows_per_partition":"NONE"}'
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCom
pactionStrategy'}
    AND compression = {'sstable_compression': 'org.apache.cassandra.io.compress.
LZ4Compressor'}
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99.0PERCENTILE';
```

```
cqlsh:twissandra> select * from users;

 username | password
----------+----------
     Joan |   joan00
    Josep |  josep00
```

```
cqlsh:twissandra> describe table users;

CREATE TABLE twissandra.users (
    username text PRIMARY KEY,
    password text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = '{"keys":"ALL", "rows_per_partition":"NONE"}'
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCom
pactionStrategy'}
    AND compression = {'sstable_compression': 'org.apache.cassandra.io.compress.
LZ4Compressor'}
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99.0PERCENTILE';
```

```
cqlsh:twissandra> select * from friends;

 username | friend | since
----------+--------+--------------------------
     Joan |  Josep | 2017-01-14 18:14:54+0000
    Josep |   Joan | 2017-01-14 18:14:27+0000
```

```
CREATE TABLE twissandra.friends (
    username text,
    friend text,
    since timestamp,
    PRIMARY KEY (username, friend)
) WITH CLUSTERING ORDER BY (friend ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = '{"keys":"ALL", "rows_per_partition":"NONE"}'
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCom
pactionStrategy'}
    AND compression = {'sstable_compression': 'org.apache.cassandra.io.compress.
LZ4Compressor'}
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99.0PERCENTILE';
```

```
cqlsh:twissandra> select * From followers;

 username | follower | since
----------+----------+--------------------------
     Joan |    Josep | 2017-01-14 18:14:27+0000
    Josep |     Joan | 2017-01-14 18:14:54+0000
```

```
CREATE TABLE twissandra.followers (
    username text,
    follower text,
    since timestamp,
    PRIMARY KEY (username, follower)
) WITH CLUSTERING ORDER BY (follower ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = '{"keys":"ALL", "rows_per_partition":"NONE"}'
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCom
pactionStrategy'}
    AND compression = {'sstable_compression': 'org.apache.cassandra.io.compress.
LZ4Compressor'}
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99.0PERCENTILE';
```

```
cqlsh:twissandra> select * From userline;

 username | time                                 | tweet_id
----------+--------------------------------------+--------------------------------------
 !PUBLIC! | 7f563944-da85-11e6-9137-000c29375491 | 35976bee-c589-41ec-a8f3-9c8d017853db
 !PUBLIC! | 70b8e008-da85-11e6-b39d-000c29375491 | cd31babf-3b7a-473f-beb6-904ca21589a8
 !PUBLIC! | e57553fa-da84-11e6-9ff1-000c29375491 | c1e3c2f3-56d8-4cf8-9c00-71d3a84492fd
     Joan | 70b8e008-da85-11e6-b39d-000c29375491 | cd31babf-3b7a-473f-beb6-904ca21589a8
     Joan | e57553fa-da84-11e6-9ff1-000c29375491 | c1e3c2f3-56d8-4cf8-9c00-71d3a84492fd
    Josep | 7f563944-da85-11e6-9137-000c29375491 | 35976bee-c589-41ec-a8f3-9c8d017853db
```

```
cqlsh:twissandra> describe table userline;

CREATE TABLE twissandra.userline (
    username text,
    time timeuuid,
    tweet_id uuid,
    PRIMARY KEY (username, time)
) WITH CLUSTERING ORDER BY (time DESC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = '{"keys":"ALL", "rows_per_partition":"NONE"}'
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}
    AND compression = {'sstable_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99.0PERCENTILE';
```

```
cqlsh:twissandra> select * from timeline;

 username | time                                 | tweet_id
----------+--------------------------------------+--------------------------------------
     Joan | 7f563944-da85-11e6-9137-000c29375491 | 35976bee-c589-41ec-a8f3-9c8d017853db
     Joan | 70b8e008-da85-11e6-b39d-000c29375491 | cd31babf-3b7a-473f-beb6-904ca21589a8
     Joan | e57553fa-da84-11e6-9ff1-000c29375491 | c1e3c2f3-56d8-4cf8-9c00-71d3a84492fd
    Josep | 7f563944-da85-11e6-9137-000c29375491 | 35976bee-c589-41ec-a8f3-9c8d017853db
    Josep | 70b8e008-da85-11e6-b39d-000c29375491 | cd31babf-3b7a-473f-beb6-904ca21589a8
```

```
cqlsh:twissandra> describe table timeline;

CREATE TABLE twissandra.timeline (
    username text,
    time timeuuid,
    tweet_id uuid,
    PRIMARY KEY (username, time)
) WITH CLUSTERING ORDER BY (time DESC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = '{"keys":"ALL", "rows_per_partition":"NONE"}'
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}
    AND compression = {'sstable_compression': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99.0PERCENTILE';
```

Introducción

- Existen muchas librerías y formas de trabajo entre Python y Cassandra
- Cassandra dispone de un driver propio para trabajar con Python, **Python-driver**, que sería el más recomendable.
- Es un manejador (‘handler’) de conexiones que trabaja a nivel de Clúster.
- El objeto clave es **session**, a través del cual se realiza la conexión.
- Se puede consultar la documentación del driver en su repositorio oficial:
  https://github.com/datastax/python-driver

Ejemplos

```
from cassandra.cluster import Cluster

cluster = Cluster()
session = cluster.connect()

session.execute("CREATE TABLE test.users (id
int PRIMARY KEY, location text)")

session.execute("INSERT INTO test.users (id,
location) VALUES (%s, %s)",
                (0, "123 Main St."))

results = session.execute("SELECT * FROM
test.users")
row = results[0]
print row.id, row.location
```

Importar depedencias

Instanciar conexión (clúster y sesión)

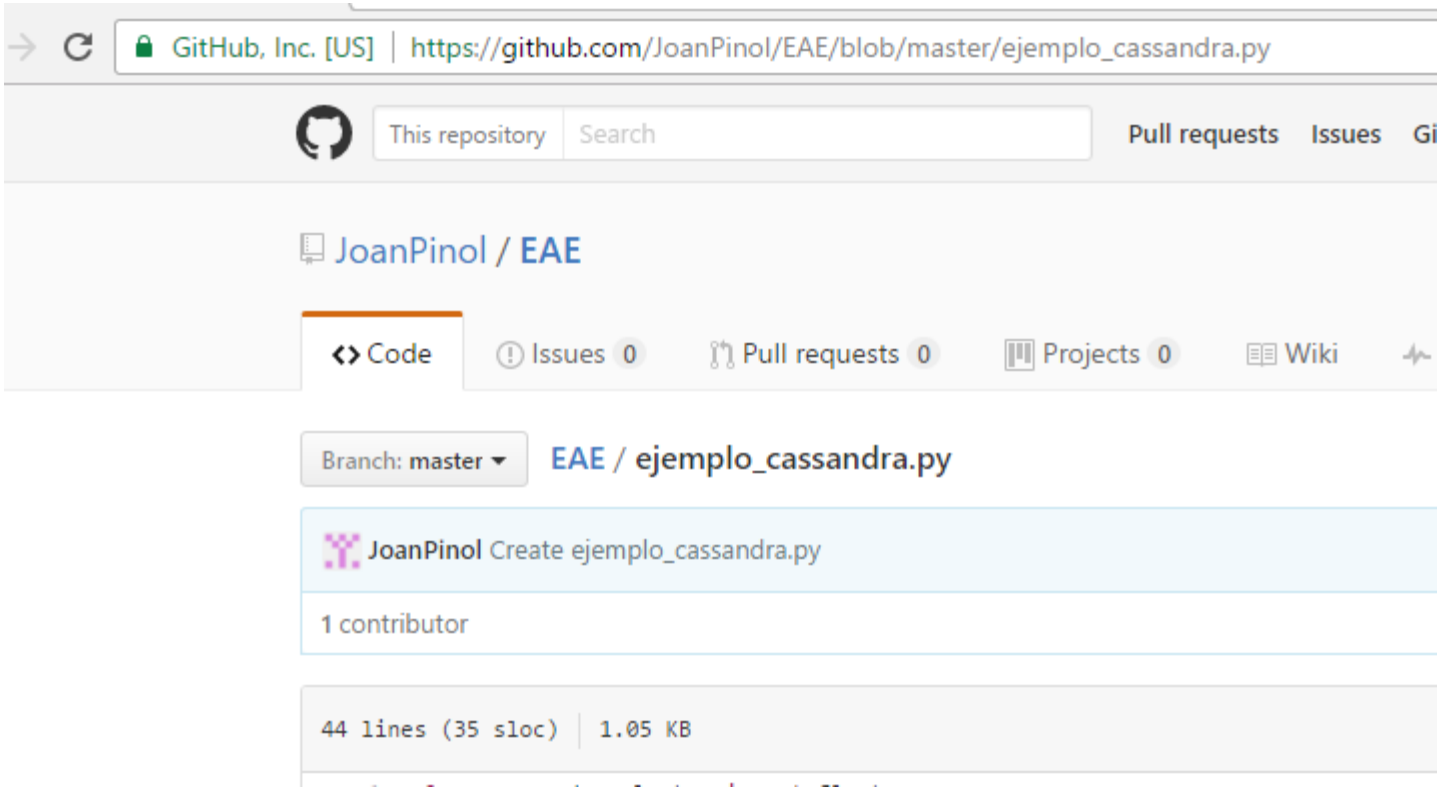
Crear tabla

Realiza insert

Ejecuta query

Muestra resultados

Ejemplos


GitHub, Inc. [US]
https://github.com/JoanPinol/EAE/blob/master/ejemplo_cassandra.py
This repository
Search
Pull requests
Issues
Gi
JoanPinol / EAE
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Branch: master
EAE / ejemplo_cassandra.py
JoanPinol Create ejemplo_cassandra.py
1 contributor
44 lines (35 sloc)
1.05 KB

Documentación Datastax

- Documentación CQL:
http://docs.datastax.com/en/archived/cql/3.0/cql/cql_using/useTOC.html

- Data modelling en 30s:
http://docs.datastax.com/en/landing_page/doc/landing_page/dataModeling.html

- Key concepts en 30s:
http://docs.datastax.com/en/landing_page/doc/landing_page/keyConcepts.html