Distribución y replicación

Distribución y replicación son los conceptos básicos que emplea Cassandra para garantizar la disponibilidad y tolerancia a fallos. En Cassandra ambas tareas se realizan do forma simultánea. Cuando un dato se distribuye, también se replica. Además de esto los datos se van a organizar en función de su clave primaria (PK). La PK (***Partition Key* no confundir con *primary key***) determina en qué nodo se van a escribir lo datos.

Elementos involucrados en la distribución y replicación

- **Nodos virtuales (*Vnodes*)**: aumentan el grado de granularidad de los datos.
- **Particionador**: determina en qué nodo se va a almacenar un dato en función de su PK.
- **Estrategia de replicación**: determina el número de copias que se van a almacenar de cada dato.
- ***Snitch***: determina la topología de la red.

Nodos virtuales (Vnodes)

Aumentan el grado de granularidad de los datos. Al comportarse como nodos *reales* permiten que un nodo almacene más datos que los que le corresponderían en una distribución sin *Vnodes*. Esto hace que haya datos replicados en más nodos y reduce el riesgo de que la caída de un nodo provoque la pérdida de datos.

El intervalo de PKs que le correspondería a un nodo se calcula a partir del valor de dos tokens. El primer token se calcula a partir del hash de la dirección IP del nodo. El segundo token se calcula a partir del hash del nombre del cluster.

Cuando añadimos un nuevo nodo, éste asume la responsabilidad sobre un conjunto de datos que le correspondía a otros nodos. Esto hace que se tenga que realizar un proceso de redistribución de datos. Este proceso se realiza de forma automática y transparente para el usuario.

La proporción de *Vnodes* por nodo es configurable.

Particionador

El particionador es el encargado de determinar en qué nodo se va a almacenar un dato en función de su PK. El algoritmo de distribución de datos utiliza una **función hash** para calcular el token de un dato a partir de su **PK**. El token es un número de 64 bits que se utiliza para determinar en qué nodo se va a almacenar el dato (en relación a los tokens del nodo). Como dijimos cada nodo tendrá dos tokens y el dado ha de almacenarse en el nodo *entre cuyos tokens* se encuentre el token del dato. Es decir, si un nodo tiene los tokens 1 y 100 y el hash del dato fuese 50 el dato se almacenará en dicho nodo.

Obviamente los tokens se generan de forma que todo el rango de posibles valores de hash de un dato esté cubierto por los rangos de tokens de los nodos.

Cassandra proporciona tres particionadores:

- Murmur 3Partitioner (por defecto).
- Random Partitioner.

- ByteOrdered Partitioner.

Estrategia de replicación

Cassandra utiliza la replicación para asegurar la disponibilidad y tolerancia a fallos. El **factor de replicación** es el valor que indica el número de copias que se van a almacenar de cada dato y **no debe sobrepasar el número de nodos del *datacenter***. El valor de esta propiedad se puede modificar en tiempo de ejecución.

La replicación se realiza de forma automática y transparente para el usuario. Cassandra proporciona dos estrategias de replicación:

- SimpleStrategy (por defecto): Usado para clusters con un único *datacenter*. Las réplicas se distribuyen en los nodos de forma secuencial.
- NetworkTopologyStrategy: Usado para clusters con varios datacenters. Las réplicas se distribuyen en los nodos de forma secuencial en función de los datacenters. Se puede definir el factor de replicación por datacenter.

Snitch

El snitch es el encargado de determinar la topología de la red. Es decir, determina a qué *datacenter* y a qué rack pertenece cada nodo. Cassandra proporciona varios snitches:

- Dynamic.
- GoogleCloudSnitch.
- Simple.
- Ec2Snitch.
- Rackinferring.
- ...ññadslafkñjbbccd

Primary key, partition key y clustering key

Una *primary key* en Cassandra estará compuesta de **una *partition key*** (simple o compuesta) **y cero o más *culstering keys***. Al definir una *primary key* la columna o columnas que conforman la ***partition key* siempre aparecerán antes** que los campos que conforman las *clustering keys*.

La *primary key* de una tabla de Cassandra puede tener la siguiente estructura:

- **Un único campo**: en cuyo caso ese campo tiene que ser único para todos los registros de la tabla. Es decir, una columna con valores únicos.
- **Dos o más campos**: en este caso lo que tiene que ser único es la combinación de los dos campos. Pueden definirse de dos formas:
  - `PRIMARY KEY (<campo1>, <campo2>, …)`: aquí el `<campo1>` será la ***partition key*** y el resto de campos serán *clustering keys*.
  - `PRIMARY KEY ((<campo1, campo2), <campo3>, …)`: en este ejemplo `<campo1>` y `<campo2>` constituyen la *partition key* y el resto serán *clustering keys* (ver **partition key compuesta**).

Partition key

La principal función de una *partition key* es la distribuir los datos de una manera uniforme entre los nodos del *cluster* y permitir consultas de una manera eficiente. Esto se llevará a cabo aplicando una **función de *hash*** a la *partition key*. Con el *hash* resultante se determinará qué nodo del *cluster* le corresponde y la **partición** dentro de dicho nodo.

Partition key simple

Si la *partition key* está formada a partir de una única columna, los valores de este campo serán los que se usen para calcular el *hash*. Una vez calculado el *hash* este determinará en que **partición** se va a guardar el registro. Al mismo tiempo **también** estamos determinando **el nodo** donde se va a almacenar ya que **todos los elementos de una partición han de estar almacenados en el mismo nodo**.

Partition key compuesta

Una *partition key* compuesta estará formada por dos o más columnas. Esto implica que se utilizarán varias columnas para determinar dónde se va a almacenar el dato. Esta técncia se utilizará **cuando la cantidad de datos a almacenar es demasiado grande para guardarse en una única partición**. Cuando usamos más de una columna en la *partition key* los datos se dividirán en *pedazos* o *buckets*. Los datos serguirán estando agrupados pero en fragmentos más pequeños. Este método puede ser efectivo en la reducción de *puntos calientes* o **congestión en escrituras**, cuando la partición de un nodo recibe gran cantidad de escrituras.

Clustering key

Clustering es el proceso de ordenar los datos de una partición y se basa en las columnas definidas como las ***clustering keys***. La definición de las columnas que serán *clustering keys* ha de hacerse con antelación. Esto ha de hacerse así ya que la elección de las columnas que serán *clustering keys* depende de cómo vamos a usar los datos en la aplicación. Es decir, la elección de las *clustering keys* tendrá un gran impacto en el rendimiento de la aplicación que consuma esos datos.

Todos los datos de una partición se almacenan de forma contigua en el dispositivo de almacenamiento ordenados según las columnas definidas como *clustering keys*. Esto hace que la recuperación de los datos sea muy eficiente (siempre que se haga respecto a estas columnas).