USING TOKENS TO DISTRIBUTE CASSANDRA DATA

Cassandra distributes data based on tokens. A **token** is the hashed value of the primary key. When you add nodes to Cassandra you assign a token range to each node, or let Cassandra do that for you. Then when you add data to Cassandra it calculates the token and uses that to figure out on which server (node) to store the new data. That works fine at the start. But as we will see, it can result in one server having lots more data than the others, as your system grows and you add nodes to the **ring** (i.e., the collection of nodes), thus resulting in an imbalance in workload. So it is necessary to give thought to how this should be configured

How Cassandra Distributes Tokens

The algorithms that calculate the token are designed to range back and forth in such a way that data is distributed evenly. You can configure token/node assignment yourself in cassandra.yaml or you can let Cassandra take a best guess estimate.

To illustrate, suppose we have this table:

```
CREATE TABLE customers.customers (
    name text PRIMARY KEY,
    credit int
)
```

and these three records:

```
select token(name), name from customers.customers;
```

```
 system.token(name)   | name
----------------------+----------------
 -1595003126634288162 |     Book Store
  2201572791825907709 |      Bok Store
  6005990370378710037 | Hardware Store
```

We can see where each of this records are stored by using the nodetool command and showing the token with the **system.token()** CQL command. The format for nodetool is **nodetool getendpoints keyspace table token**.

As we can see from the ip addresses below, in our cluster of two servers the records are distributed across both.

```
nodetool getendpoints customers customers -1595003126634288162
172.31.47.43
nodetool getendpoints customers customers 2201572791825907709
172.31.46.15
nodetool getendpoints customers customers 6005990370378710037
172.31.47.43
```

The Problem with Uneven Data Distribution

Now, if you stop and think about this, that works fine when you first set up your cluster. But as your system grows Cassandra can pile up data on one node and leave others underutilized. Plus there is the problem of how to move existing data onto new nodes. So what do you do?

There are three different algorithms you can use to assign tokens to nodes. **Murmur3Partitioner** in cassandra.yaml lets you set **allocate_tokens_for_keyspace.** You can use that with **num_tokens** to override the random distribution of tokens (meaning the other partitioner algorithms). This will let new nodes take more of the data from existing nodes thus reducing the over/under utilisation problem. It does this by letting the assignment of ranges to nodes shift dynamically based upon data load in the other nodes. But since it does not move existing data over to the new nodes will take some time for the data to be spread move evenly.

num_tokens means the division of a node into virtual node. The default is 256.

In the next post we will explain in more detail this algorithm so you can figure out how to tune this for your environment.