DATASTAX

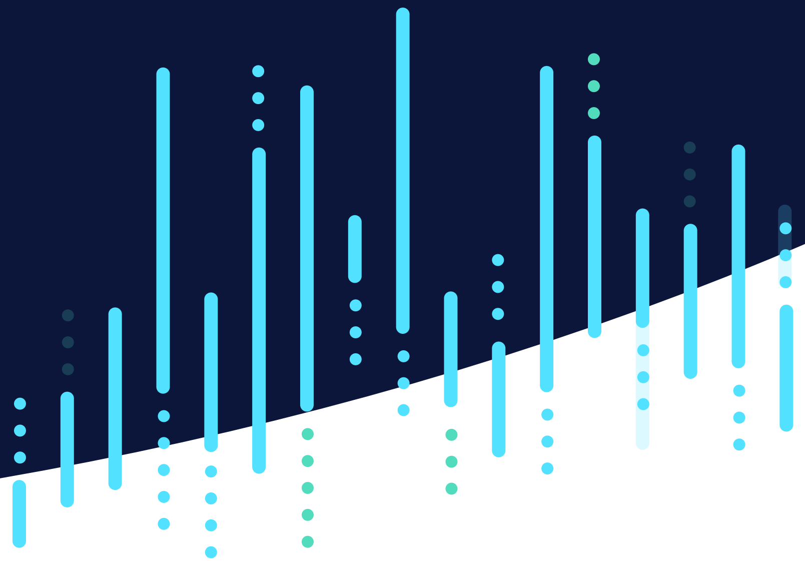
Back to Blog

Consensus on Cassandra

DATE: December 1, 2014

AUTHOR:

Jake Luciani
DataStax



Consensus, the agreement among peers on the value of a shared piece of data, is a core building block of Distributed systems. Cassandra supports consensus via the paxos protocol since 2.0. We often see out in the field folks using tools like ZooKeeper, etcd, and even redis!? to provide consensus among peers for different parts of their system. If you are using Cassandra and want to avoid adding Zookeeper, I'm going to show a simple approach solving a common use case for consensus: providing a leader election algorithm on top of Cassandra using Paxos.

The Data Model:

First let's create a lease table to track owners of things. A lease can be used to provide consensus on shared resources/setting and provides a mental model that's easy for developers to understand. A good read on this topic is Google's chubby paper. ***Note***, this approach assumes there are no bad actors in the system executing non standard CQL to the table. One way to ensure this is to use access controls on the table.

```
CREATE TABLE leases (
      name text PRIMARY KEY,
      owner text,
      value text
 ) with default_time_to_live = 180
```

Simple enough. A lease has a name, optional value and an owner. For clarity we'll keep the types as text. There is also a default TTL of 3 minutes. This TTL is used as a fail-safe to avoid deadlocking on a resource if an owner dies before revoking the lease. This also gives us some notion of ephemeral values.

A client acquires a lease by issuing the following statement:

```
cqlsh:dev> INSERT INTO leases (name, owner) VALUES ('foo','client_unique_id_1') IF NOT EXISTS;


[applied]
 -----------
 True
```

If the lease already exists and is held by someone else we get back the existing data.

```
[applied]  | name | owner              | value
-----------+------+--------------------+-------
 False     | foo  | client_unique_id_1 | null
```

Once we have acquired the lease we know we have it for a max of 3 minutes before it's revoked. So we must periodically renew/KeepAlive the lease using the following CQL.

```
cqlsh:dev> UPDATE leases set owner = 'client_unique_id_1' where name = 'foo' IF owner = 'client_unique_id_1';

[applied]
 -----------
 True
```

If the lease was expired before the renewal happened and some new owner had leased it, the IF clause protects us from breaking protocol. Once we no longer need the lease we can explicitly drop it with the following CQL:

```
cqlsh:dev> DELETE FROM leases where name = 'foo' IF owner = 'client_unique_id_1';
[applied]
 -----------
 True
```

If, say, the client was garbage collecting and the lease expired before the delete was sent we avoid deleting someone else lease with this owner check.

Finally, to get the existing value of the lease we can read it with the following CQL:

```
cqlsh:dev> select writetime(owner), value, owner from leases where name = 'foo';


writetime(owner)  | value | owner
------------------+-------+--------------------
 1417455697411000 | null  | client_unique_id_1
```

Note, this needs to be executed with consistency level of SERIAL to ensure we get serialized view of the data.

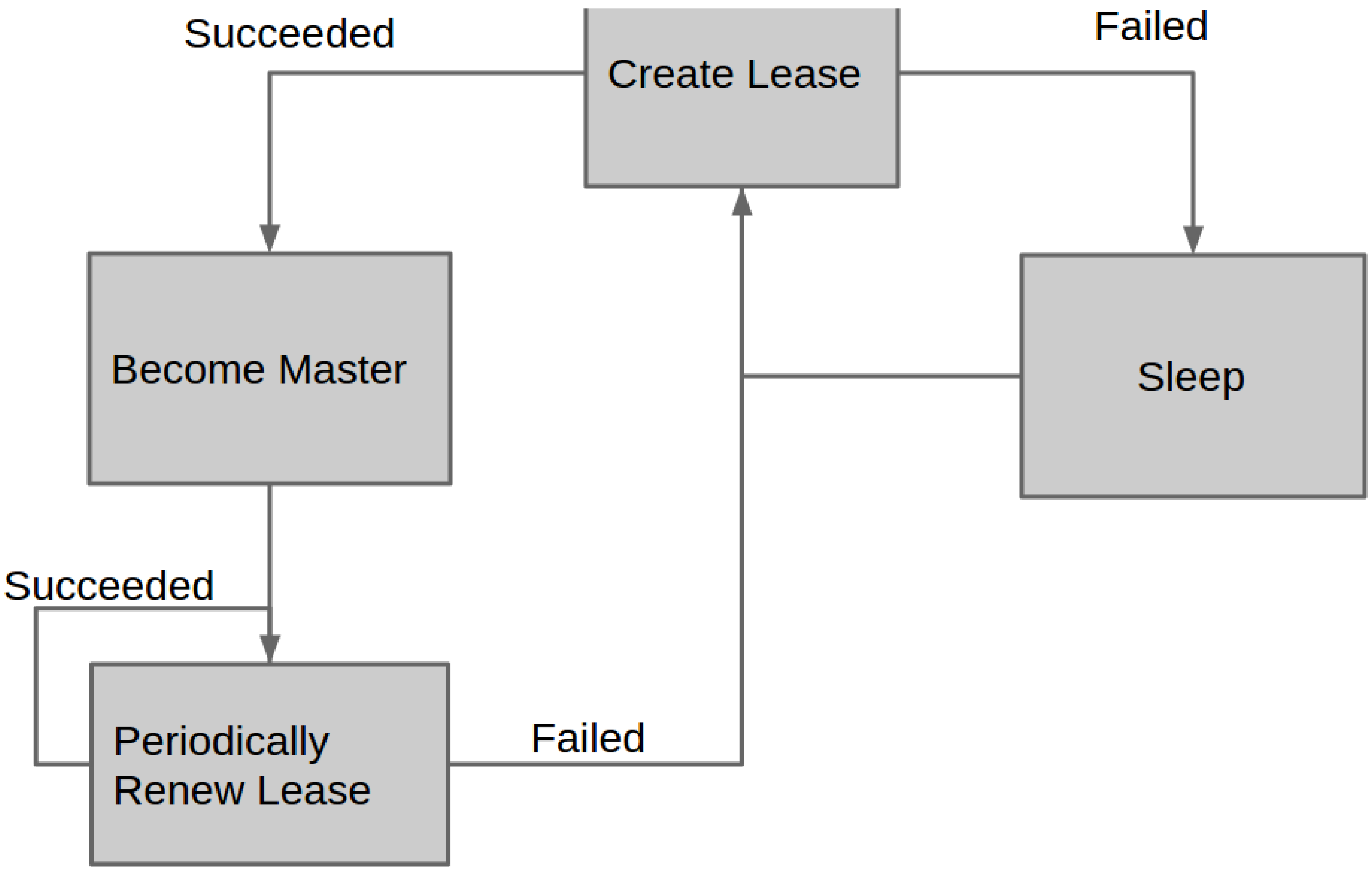
The write timestamp returned from this query can be used by clients to figure out the time they should wait before polling again (take write time and add TTL time in milliseconds).

We didn't use the value column in this example but this column could be used to store things like the ip address and port number of the elected service.

Leader Elections:

We can elect leaders of things with this approach by having nodes that want to become master try to grab the lease on start. Cassandra will return the same result to all clients and the one that was elected master will renew the lease in the background every N seconds. The clients will poll to verify the master is still master.

If the master dies the clients will see the lease missing and wait for a new leader to be elected by the remaining participants.



Other types of consensus:

Using techniques like this it is also possible to build things like distributed locks and distributed sequences on top of Cassandra. Google's chubby paper is, again, a good resource on how to design this kind of system.

DATASTAX

Whitepapers
Reports
Videos
Podcasts
Blog

Instructor Led Training
Public Training
Developer Blog

DataStax Bulk Loader
DataStax Apache Kafka® Connector
CQLSH
DataStax Drivers
DataStax Labs

General Inquiries: +1 (650) 389-6000 info@datastax.com