written by Wojtek Wajerowicz on 16 September 2015 in Cassandra, engineering

thrift to CQL

As you all know, we're using Cassandra as our data store in Atlas Deer. Atlas Deer has been in development for quite some time, but we've only recently started to use it seriously in production. Cassandra has also developed quite a bit during this time, therefore it's time to revisit.

Currently our schedule store is using the Thrift interface for Cassandra. However, nowadays CQL (Cassandra Query Language) is the recommended way of interacting with Cassandra.

CQL is a language which abstracts away the internal storage model of Cassandra which enables you to write queries more easily.

Currently we store schedules as follows: we use the source of the schedule, channel ID and day as the row key and one column called `IDS` which stores ids of current broadcasts in the schedule (we need to be able to keep track of past schedules to be able to delete stale entries in equivalent schedules). There's also a column with broadcast id as a name and serialized broadcast as value for each broadcasts. An example row for a schedule would look like:

IDS	BROADCAST_1	BROADCAST_2
BROADCAST_1,BROADCAST_2	serialized broadcast	serialized broadcast

In CQL this could be replaced by:

```
CREATE TABLE schedule_v2 (
   source text,
   channel bigint,
   day timestamp,
   broadcast_ids set<text>,
   broadcasts map<text,blob>,
   updated timestamp,
   PRIMARY KEY ((source, channel, day))
);
```

CQL is a nicer and easier to use way of talking to Cassandra. It makes querying Cassandra more human readable and enables you to write SQL-like queries . However, there's one important caveat: in order to write efficient CQL queries, you still need to understand underlying storage model and how CQL maps to it. Otherwise, you'll try to write SQL queries which won't be understood by Cassandra or, even worse, queries which will be extremely inefficient. For people who already

understand the Cassandra row model, I highly recommend Robbie Strickland's presentation about how CQL maps to underlying storage.

If you enjoyed the read, drop us a comment below or share the article, follow us on Twitter or subscribe to our #MetaBeers newsletter. Before you go, grab a PDF of the article, and let us know if it's time we worked together.