written by Sergio Bossa on 13 July 2012 in Atlas, Cassandra

looking with cassandra into the future of atlas

It's no surprise we use MongoDB almost everywhere.

And it's no surprise either that we're planning for world domination, ingesting more and more data from different media sources into our Atlas platform: which is leading us to hit MongoDB scalability limits, discover obscure bugs, and question some of its design choices.

That's why for the upcoming Atlas 4.0 version we're going to switch gears and move towards one of the most dominant players in the non-relational databases space: Cassandra.

is it because you're all crazy hipsters?

The reason we're moving from MongoDB to Cassandra is not because we love the latest and coolest technology, whatever that means. Rather, given the large growth we're experiencing on the Atlas platform in terms of data and users:

- We need higher resiliency to faults: MongoDB provides replica sets, but we're experiencing lots of problems with replication lags, and during replica synchronization
- We need higher scalability: MongoDB global lock and huge memory requirements aren't already going to cope well with our growing data set

So, we chose to go with Cassandra for a number of reasons:

- It works on the JVM, and we have lots of in-house experience on it
- It scales in terms of processing and storage capacity
- Its column-based data model gives us some advanced capabilities we will talk about in a few minutes
- Its tunable consistency levels provide greater control over high availability and consistency requirements

But we didn't stop at mere theoretical considerations: once settled on Cassandra, it was time to do some prototyping and testing.

We deployed a Cassandra cluster on AWS, prototyped a schema for our data and started testing with CassJMeter (to which we even contributed).

Tests passed with flying colours. It was time to welcome Cassandra into Atlas 4.0 and do some serious work with it: let's review a few challanges we had to face.

first things first: organizing data in a schemaless world

Cassandra is very different from your relational database of choice. Also, Cassandra is a kind of schemaless database, but with its own data model: you have to define a keyspace and its column families, but you can have as many columns as you want inside your families, which gives you lots of flexibility… and uncertainity about how to organize data.

Our data is made up of entities with several fields and nested entities; in order to model it into Cassandra, we could have either gone by:

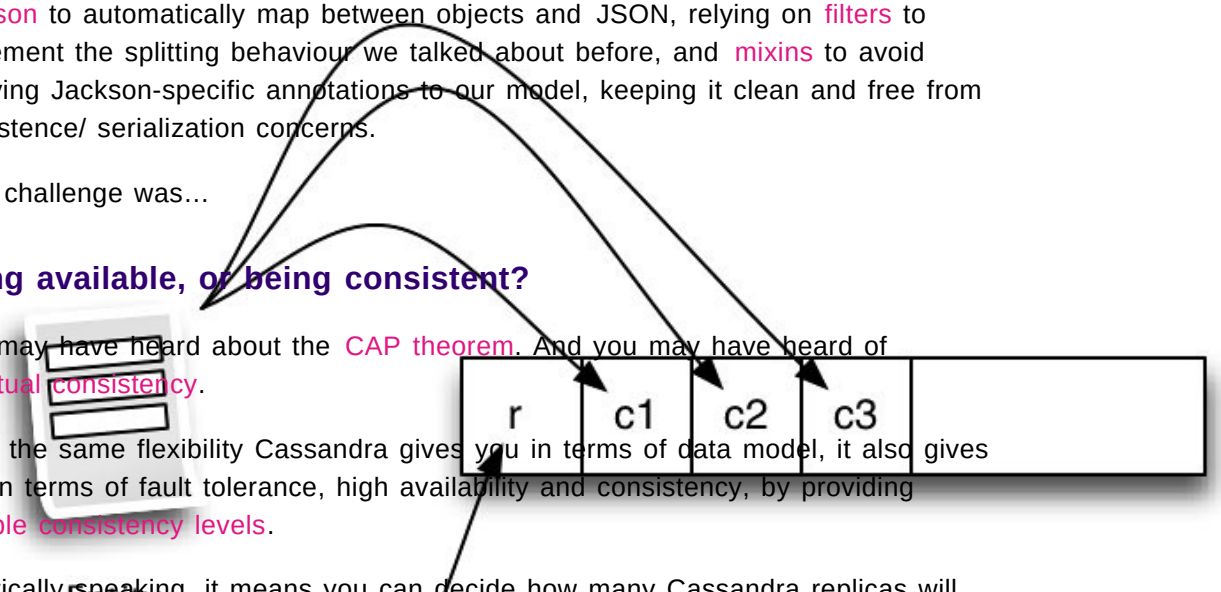
- Modeling our entities as opaque values, actually using Cassandra as a pure key/value store
- Or mapping each entity field to a different column and establishing by-id relations between entities, practically using Cassandra as a relational database

We chose instead to model our data based on access patterns: our entities are often huge, and different clients often need to access different parts at different times, so we store each entity by splitting it in different parts, each one in its own column. While right now we're still in full development, this will allow us to only retrieve entity parts the client is actually interested in, so cutting network bandwidth and reducing memory footprint by the server and client side.

Implementation-wise, entities are stored inside Cassandra in JSON format: we use Jackson to automatically map between objects and JSON, relying on filters to implement the splitting behaviour we talked about before, and mixins to avoid applying Jackson-specific annotations to our model, keeping it clean and free from persistence/ serialization concerns.

Next challenge was…

being available, or being consistent?



You may have heard about the CAP theorem. And you may have heard of eventual consistency.

Well, the same flexibility Cassandra gives you in terms of data model, it also gives you in terms of fault tolerance, high availability and consistency, by providing tunable consistency levels.

Practically speaking, it means you can decide how many Cassandra replicas will answer your write and read requests, making a tradeoff between high availability (fewer nodes answering to avoid being let down by failing nodes) and strong consistency (more nodes answering to increase chances of an up-to-date answer).

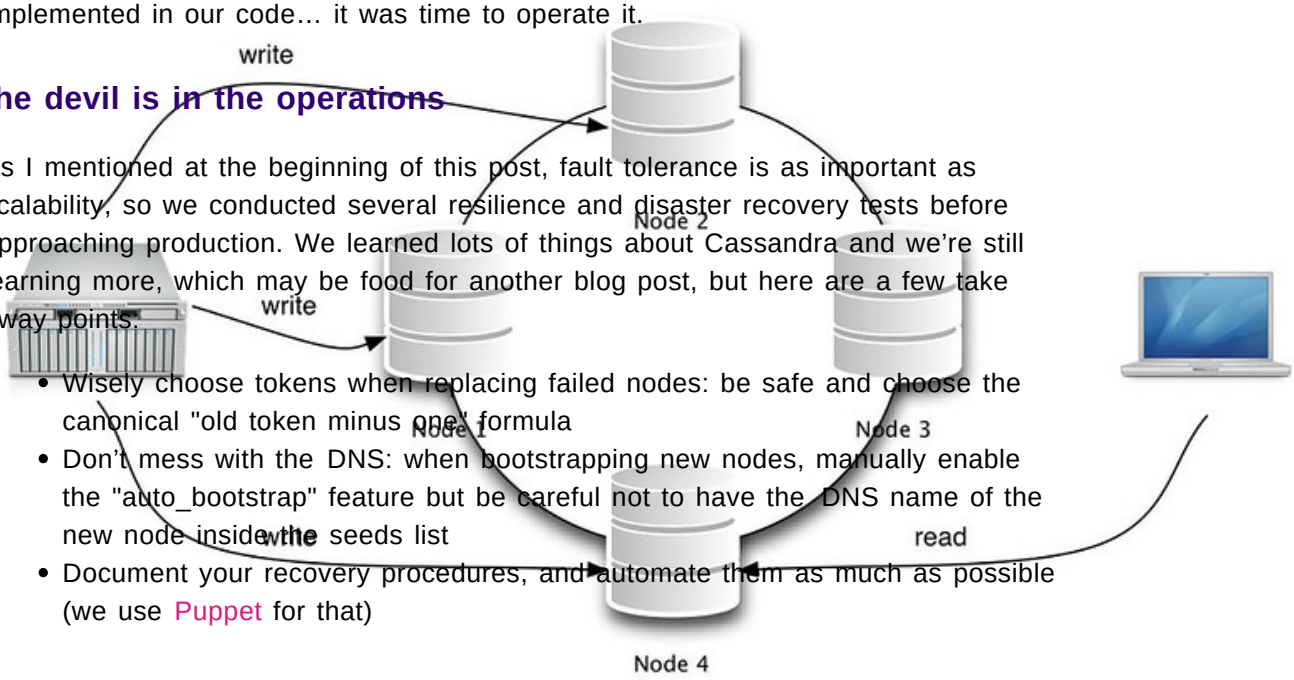
Again, our choice was driven by our actual requirements.

Our Atlas platform is made up of several background processes ingesting data from different sources: here, data consistency is paramount, so we chose to go with "quorum" consistency, meaning that a majority of replica nodes needs to successfully acknowledge write requests so as to avoid inconsistencies.

On the other side, we have a huge number of clients reading data from the platform: here, availability and performance are paramount, so we chose to go with "one" consistency level, meaning that we only wait for a single answer to our read

requests, even if it may lead to some stale data (which will be automatically repaired later).

So we had our data model and access patterns crystal clear in our minds, and implemented in our code… it was time to operate it.



the devil is in the operations

As I mentioned at the beginning of this post, fault tolerance is as important as scalability, so we conducted several resilience and disaster recovery tests before approaching production. We learned lots of things about Cassandra and we're still learning more, which may be food for another blog post, but here are a few take away points:

- Wisely choose tokens when replacing failed nodes: be safe and choose the canonical "old token minus one" formula
- Don't mess with the DNS: when bootstrapping new nodes, manually enable the "auto_bootstrap" feature but be careful not to have the DNS name of the new node inside the seeds list
- Document your recovery procedures, and automate them as much as possible (we use Puppet for that)

the end?

We release early and release often, so I'm proud to announce we just rolled out Cassandra into production for Atlas 3.0, too, serving the brand new music data—yes, our global video & audio index is going to ingest and serve music metadata, too, for a rounder meaning of audio!

Of course, we're still working on it: large part of our data is still in MongoDB, and we're going to migrate it in the next couple of months, while also finalising all the cool features we've talked about before for Atlas 4.0 (more to follow).

And since Cassandra doesn't provide query capabilities as rich as MongoDB (the only missing feature) we decided to implement with ElasticSearch: food for another blog post again. In the meantime, hope you enjoyed this one!