Data Driven Organizations, Case Study: Craigslist – Avi Skidelsky

Craigslist is one of the most frequented e-commerce websites in the world, ranking #1 in that category in the US and had nearly 210 million visits in the past month alone. However, when they were first founded they could not have imagined the success that they would have and because of this, they were not well equipped to handle the amount of data that they would be generating and continually using.

Craigslist’s initial database was a standard relational database. It was structured so that there was a backend database and a front-end database, called the “archive” and “live” respectively, that were interactive. The archive was an exact replica of the live so that a listing would exist in the live database and after a period of time it would be relocated to the archive in an identical format. This was done so that if the archived listing needed to be referenced then it could be done so in the same format as if it were live in an effort to simplify queries.

Unfortunately, using this approach led to the schema being nearly impossible to use. As opposed to the live database which remained a similar size no matter the time, the archive kept growing and growing. This was because the live database had a revolving door of listings, listings would be entered and listings would be removed. On the other hand, the archive only ingested data to be the backend and exploded in size. It had gotten to the point that if they ever needed to make any sort of database change, even something as simple as an ALTER statement to make a change on a table it could take up to a month to run the query. In addition to the lack of performance using this method, it relied on an outdated physical system and like all hardware, it would occasionally break down and someone would need to go to the data center and repair it. Given that it took a month to run an ALTER statement, there’s no telling how long rebooting and restoring the entire system would take. Additionally, due to the constant growth of the archive they had to keep upgrading the hardware to meet their needs and that proved to be expensive and exhaustive.

All of this led to Craigslist moving on from a RDBMS model to a NoSQL model. They opted to go with MongoDB, an open-source document-oriented database program. Thanks to this new model, which was initially intended to encompass five billion documents and ten terabytes of data, each listing could live on its own not in relation to every other listing, they were able to optimize performance and efficiency.

There were plenty of lessons to be learned from this experience:

* Know your hardware: in the Craigslist production environment, every service was on a different server dedicated to that specific action so they were able to get a better handle on controlling everything. In the development environment though, Mongo didn’t cooperate with multiple tasks being run at the same time.
* Replica sets rock: by creating replica sets of what was working until a restart was needed they were able to set up a sort of “jumping off point” where they could restart at that point, already having transferred a portion of the data without any problems. This would be even better done by using some of the AWS services that allow for temporary, small batches of the data to be saved temporarily.
* Know your data very well: the data that they had wasn’t in a uniform format, specifically UTF-8 which Mongo runs on. This caused a lot of troubleshooting while importing and reformatting every time the data form wasn’t compliant.
* Know your data size: Mongo had their own size constraints and even though the files in the Craigslist database typically didn’t violate those constraints, and most files won’t, there are outliers that need to be addressed and by knowing the constraints and applying that information to your data beforehand you can save steps of going back and reformatting.
* Knowing your data type: storing isn’t the biggest issue when it comes to conflicting data types and misclassification, but querying is where that becomes an issue, causing issues searching for the wrong information.
* Know sharding: even though there are balancers in place to do this for you, it’s helpful and possible to know what should be split into which shards so that it’s in the exact format that the end user would find it useful.