# Nathan Milford on Cassandra

## An Interview

RIK FARROW

Nathan Milford (@NathanMilford) is the US Operations Manager at Outbrain. Nathan is interested in large data projects, scalable architectures, and open source. In his spare time, he is a photographer and practitioner of Jiu Jitsu and Muay Thai. You can follow his blog at http://blog.milford.io/.
nathan@milford.io

Doug Hughes introduced me to Nathan Milford when he learned that I was looking for someone who could talk about his experience using Cassandra. Before talking with Nathan, I read his excellent slide deck about working with Cassandra [1], and watched part of his presentation of these slides [2] from the Cassandra NYC conference last December.

After a short phone conversation over the noise in his datacenter, Nathan agreed to continue our talk by email.

**Rik:** Could you tell us a little about what Outbrain does, to provide us with the background we need to understand why you chose to use Cassandra?

**Nathan:** I'll just hit you with what my marketing team would have me say:

> Outbrain is the leading content discovery platform, helping publishers, brands, and agencies reach a highly engaged audience through distribution on leading media sites. Outbrain works with publishers like CNN, Fox News, Hearst, Rolling Stone, and MSNBC as well as brands and agencies, including American Express, P&G, General Electric, Media Contacts, and Starcom to increase site traffic and generate new revenue through customized links to recommended content.

In short, we're a content discovery and recommendation engine. We've got dozens of paid and organic recommendation algorithms that dig into our Hadoop, Solr, Cassandra, and other clusters and return, not only other content that is like what you're reading, but other content that will likely be interesting to you.

**Rik:** That does sound interesting, but could you provide more detail?

**Nathan:** Gladly. We use Cassandra as a persistent cache of calculated recommendations.

The (somewhat simplified) flow for our operation goes something like this:

- A user opens up a an article on, say, CNN.com.
- Our widget loads from a CDN and pings one of our three datacenters with the site and document IDs.
- A bevy of Tomcat instances behind HAProxy grab the document info, then query Memcached looking for pre-calculated recommendations for that document.
- If Memcached doesn't have it, Tomcat will ping another app we call the CacheWarmer.
- If it is a new document, the CacheWarmer will send a request into ActiveMQ (a commonly used queue and message broker) to have various offline processes

crawl, index, and calculate the recommendations for it. This process can (depending on the algorithms involved) hit Solr, Hadoop/Hive, MogileFS, MySQL, Cassandra, and/or a bunch of other internal processes that our brilliant R&D teams have concocted.

- The calculated recommendation data is then put into Memcached, where it will eventually expire, and into Cassandra, where it lives for much longer but also will eventually expire, thanks to Cassandra's TTL feature.
- If it is a document we know about, we hit up Cassandra for it and float it into Memcached.

We're also building other Cassandra clusters for other uses. We have a large document mapping table in MySQL that is essentially a key/value store, and a good fit for Cassandra's data model.

Before we started using data stores other than MySQL, we had a single-master MySQL setup with slaves distributed across datacenters. Since we're read-heavy, it makes sense. However, data and traffic keep on growing, and fixing replication issues and managing a brittle topology requires more and more attention.

When not all of your data needs the features MySQL offers, you come to a place where you weigh the advantages of federating your data out into appropriate data stores and having to manage a menagerie of newfangled systems versus fitting all your data into MySQL and dealing with the feature overhead and keeping a system everyone already knows.

It's not for everyone, but we chose to use the menagerie.

**Rik:** Why did you choose Cassandra over other NewSQL databases? Were others in the running?

**Nathan:** We started using TokyoTyrant on SSDs, but at the time the project had a small community, the developer was not always responsive, and it was a bit unpolished operationally. It was not crash-safe, and managing replication was a challenge sometimes, in that the mechanism was pretty basic.

We looked into HBase, but we were turned off by the HDFS append patches you needed to mess with at the time (it has since gotten better and more reliable). Also, we wanted something that would reduce operational complexity, so running multiple Hadoop clusters just to run HBase on top of it as well as keeping the clusters in sync seemed like a lot of work.

Cassandra hit the sweet spot for performance and operational complexity. Dealing with replication across multiple datacenters is pretty trivial.

The biggest difficulty is getting people to model data for it properly and not treat it like MySQL. Once you model the data and have a query plan that suits it, Cassandra is pretty hands-off from an operational perspective.

We've been using Cassandra in production since version 0.5.0 (1.1 was just released). We've had some rough patches, but nothing wildly discouraging, and, for the most part, it just works.

Since 0.5.0 we've gotten a SQL-like query language called CQL, JDBC drivers, rolling upgrades, live schema management, encryption, compression, TTLs, secondary indexes, distributed counters, pluggable everything, performance parity between reads and writes, and a wildly long list of other great work by all the committers and community.

I think one of Cassandra's strongest attributes is the Cassandra community, which is very open and accepting of even people with the smallest, most trivial use cases. You can even get commercial support from the guys at DataStax, all of whom are pretty sharp folks.

You also have large players like Twitter and Netflix using it. [Ed.: Netflix is moving away from Oracle to use Cassandra exclusively [3, 4].] Netflix actually showed how linearly it scales by doing a stress test scaling from 0 to 288 nodes in EC2 [5].

**Rik:** When we talked earlier, you mentioned having a LAN party where everyone got Cassandra set up and running in less than 20 minutes. Is it really that easy?

**Nathan:** Yes. I am one of the organizers of the NYC Cassandra Meetup group along with Ed Capriolo, Jake Luciani, Levon Lloyd, and Eric Tamme. Ed had a wonderful idea where we'd have everyone bring a laptop (Windows, Mac, and Linux) with a recent JVM. We divided people into three groups, told them to plug into a different switch representing a different "datacenter," and had them install the Cassandra binary package.

The hardest part was herding everyone into the respective areas and then onto the network. It took ~30–40 minutes to get everyone set up with the right network settings and maybe 10 minutes after that to get everyone on the cluster. Shortly after that we were inserting a key in "New York" and watching it replicate to "France" and "Tokyo" [6].

Cassandra is pretty complex, but the majority of that complexity exists to keep you from having to worry about its complexity.

I do a talk on how easy it is and what a boon it is to not have to deal with replication and repair and other administrative junk.

**Rik:** How did you size up your requirements, that is, the number of Cassandra nodes you needed for your application?

**Nathan:** We were not very scientific about it when we started 2–3 years ago. We do 30 billion impressions a month, about a billion a day. You can do all the speculation and math and planning in the world, but at that scale, you just need to put traffic on it and let it sink or swim.

For the most part we found it to be a good swimmer.

Ultimately, the first iteration of our cluster was just some spare nodes. Over time our data, our traffic, and Cassandra's performance profile changed and we migrated to new hardware while we played with different file systems, disk configs, row and key caching, heap sizes, garbage collectors, etc.

Our current hardware spec is in the slide deck [1], but we have more nodes and are running 1.0.7 now.

**Rik:** How difficult is it to add new nodes?

**Nathan:** It is not difficult really, but it is a reasonably manual process. You need to recalculate your ring [7], then start up the new nodes with the correct token (which defines what part of the ring they own), then move each node's token and let them shuffle the data.

It is all a background process and is only limited by your bandwidth. If you have a multi-datacenter cluster, and slow transport between them . . . well, you'll just have to wait.

Cassandra is not for every use-case and certainly not for every type of data, but all in all, I'm very happy we went with it. It fits a nice niche in our environment and the community around it is a joy to participate in.

### *References*

[1] Nathan Milford's slide deck on Cassandra: http://www.slideshare.net/nmilford/cassandra-for-sysadmins.

[2] Milford's video on using Cassandra: http://blog.milford.io/2012/01/cassandra-nyc-2011-talk-on-youtube/.

[3] Adrian Cockcroft on Migrating from Oracle to Cassandra: http://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra.

[4] Interview with Adrian Cockcroft in *;login:*: https://www.usenix.org/publications/login/february-2012/netflix-heads-clouds-interview-adrian-cockcroft.

[5] Adrian Cockcroft and Denis Sheahan, "Benchmarking Cassandra Scalability on AWS": http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html.

[6] Cassandra/LAN party: http://www.datastax.com/dev/blog/cassandra-nyc-lan-party; http://www.edwardcapriolo.com/roller/edwardcapriolo/entry/cassandranyc_nosql_lan_party_was.

[7] Rebalancing after adding a node: http://blog.milford.io/2011/05/in-which-i-discourse-on-java-bloat-and-cassandra-node-balancing/.