The State of NoSQL in 2012

Tuesday, January 24, 2012 at 9:15AM

Todd Hoff in nosql

This is a guest post by Siddharth Anand, a senior member of LinkedIn's Distributed Data Systems team.

Preamble Ramble

If you've been working in the online (e.g. internet) space over the past 3 years, you are no stranger to terms like "the cloud" and "NoSQL".

In 2007, Amazon published a paper on Dynamo. The paper detailed how Dynamo, employing a collection of techniques to solve several problems in fault-tolerance, provided a resilient solution to the on-line shopping cart problem. A few years go by while engineers at AWS toil in relative obscurity at standing up their public cloud.

It's December 2008 and I am a member of Netflix's Software Infrastructure team. We've just been told that there is something called the "CAP theorem" and because of it, we are to abandon our datacenter in hopes of leveraging Cloud Computing.

Huh?

A month into the investigation, we start wondering about our Oracle database. How are we are going to move it into the cloud? That's when we are told that we are to abandon our RDBMS too. Instead, we are going to focus on "AP"-optimized systems. "AP" or high-availability systems is our focus in 2008-2010 as Netflix launches its video streaming business. What's more available than TV right? Hence, no downtime is allowed, no excuses!

Fast forward to the end of 2011: the past 3 years have been an amazing ride. I helped Netflix with two migrations in 2010: the first was from Netflix's datacenter to AWS' cloud, the second from Oracle to SimpleDB. 2011 was no less exciting: with a move from SimpleDB to Cassandra, Netflix was able to expand overseas to the UK and Ireland. Since Cassandra offers configurable routing, a cluster can be spread across multiple continents.

Fast forward to today: I've completed my first month at LinkedIn. I've spent this month getting familiar with the various NoSQL systems that LinkedIn has built in-house. These include Voldemort (another Dynamo-based system), Krati (a single-machine data store), Espresso (a new system being actively developed), etc… LinkedIn is now facing similar challenges to the Netflix of 3 years ago. Traffic is growing and both Oracle and the datacenter face potential obsolescence.

NoSQL and Cloud Computing to the rescue?

The State of NoSQL Today

Ignoring the datacenter vs. public cloud question for the time-being, what would I pick today regarding a NoSQL alternative to Oracle? Not many people get a chance to solve the same tough problem twice.

For one, there are still quite a few NoSQL alternatives in the market. Some are supported by startups (e.g. Cassandra, Riak, MongoDB, etc..), some are supported indirectly by companies (e.g. LinkedIn's support of Voldemort, Facebook & StumbleUpon's support of HBase), and some are supported directly by companies (e.g. AWS's S3, SimpleDB, and DynamoDB, etc…).

In making a decision, I'll consider the following learnings:

- Any system that you pick will require 24-7 operational support. If it is not hosted (e.g. by AWS), be prepared to hire a fleet of ops folks to support it yourself. If you don't have the manpower, I recommend AWS' DynamoDB
- Just because the company got by with one big machine for Oracle, don't be surprised if the equivalent NoSQL option results in 36 smaller machines. All complete solutions to fault-tolerance support "rebalancing". Rebalancing speed is determined by data size of a shard. Hence, it's better to keep the size per shard reasonable to minimize MTTR in times of disaster.
- Understand the limitations of your choice:
  - MongoDB, at the time of this writing, has a global write-lock. This means that only one write can proceed at a time in a node. If you require high write-throughput, consider something else
  - Cassandra (similar to other Dynamo systems) offers great primary key-based access operations (e.g. get, put, delete), but doesn't scale well for secondary-index lookups
  - Cassandra, like some other systems, has a lot of tunables and a lot of internal processes. You are better off turning off some features (e.g. Anti-entropy repair, row cache, etc…) in production to safeguard consistent performance.

Many of the NoSQL vendors view the “battle of NoSQL” to be akin to the RDBMS battle of the 80s, a winner-take-all battle. In the NoSQL world, it is by no means a winner-take-all battle. Distributed Systems are about compromises.

A distributed system picks specific design elements in order to perform well at certain operations. These design elements comprise the DNA of the system. As a result, the system will perform poorly at other operations. In the rush to win mindshare, some NoSQL vendors have added features that don’t make sense for the DNA of the system.

I’ll cite an example here. Systems that shard data based on a primary key will do well when routed by that key. When routed by a secondary key, the system will need to “spray” a query across all shards. If one of the shards is experiencing high latency, the system will return either no results or incomplete (i.e. inconsistent) results. For this reason, it would make sense to store the secondary index on an unsharded (but replicated) system. This concept has been utilized internally at Netflix to support internal use-cases. Secondary indexes are stored in Lucene to point to data in Cassandra.

LinkedIn is following the same pattern in the design of its new system, Espresso. Secondary indexes will be served by Lucene. The secondary index will return rowids in the primary store.

This brings me to another observation. In reviewing Voldemort code recently, I was impressed by the clarity and quality of the code. In core systems, whether distributed or not, code quality and clarity goes a long way. Although Voldemort is an open source

project and has been for years, a high degree of discipline has been maintained. I was also impressed by the simplicity of its contract - get(key), put(key,value), and delete(value). The implementors understood the DNA of the system and did not add functionality to the system that is ill-suited to the DNA.

In a similar vein, Krati, Kafka, Zookeeper, and a few other notable open-source projects stick to clear design principles and simple contracts. As such, they become reusable infrastructure pieces that can be used to build an Distributed System that you need. Hence, the system we end up building might be composed of several specialty component systems that can be independently tuned, in some ways similar to HBase. As a counter example, to achieve predictable performance in Cassandra without a significant investment in tuning, it may be easier to turn off features. This is because multiple features in a single machine contend for resources — since each feature has a different DNA (or resource consumption profile), performance diagnosis and tuning can be a pain point.
