Migrating to Cassandra in the Cloud, the Netflix Way

Jason Brown - @jasobrown

Senior Software Engineer, Netflix
Tech History, 1998-2008

- In the beginning, there was the webapp
  - and a single database
  - in a single datacenter
- Then Netflix grew, and grew, and grew
  - More databases, all linked
  - Database links, PL/SQL, materialized views
  - Multi-Master Replication (MMR)
History, 2008

- Then it melted down (Aug 2008)
  - Hardware driver
  - “one in a billion chance”
- Couldn’t ship DVDs for ~5 days
History, 2009

- Time to rethink everything
  - Abandon our datacenter
  - Ditch the monolithic webapp
  - Migrate SPOF database

- On-demand streaming was becoming the thing!
SimpleDB
  Managed by Amazon
  Got us started with NoSQL in the cloud
Problems:
  High latency, rate limiting (throttling)
  (no) auto-sharding, no backups
  We were running at 10x the intended capacity
History, 2011-now

- Cassandra
  - Similar to SimpleDB, but with limits removed
  - Dynamo-style, master-less system
  - Great multi-datacenter support
  - Written in Java
Cassandra in 3 minutes

- Dynamo and Big Table papers
- CAP Theorem: AP
  - Eventually consistent
- Distributed hash table
  - Each node takes range on ring
- Replication
  - RF = how many copies to keep
  - Within datacenter (shard across nodes)
  - Across datacenters (full data set)
- Peer-to-peer
  - Gossip
  - Failure detection
Anti-Entropy protocols

- Hinted handoff
- Read repair
- Node repair
Writes

- mmap’d commit log files
- Mutations buffered in memtable
- Flushed to immutable files (sstable)

Reads

- Check all sstables for key
- Bloom filter used for IO optimization
Compactions

- Several styles, but basically reduce the number of sstables by merging data
- Eliminate expired/deleted data
- I/O & CPU intensive
- Tunable Consistency
  - 1, 2, Quorum (local, each), All
- Columnar, Key-Value storage
  - Schema-less
  - Denormalization (wide rows)
- Cassandra Query Language (CQL)
Cassandra at Netflix
## By the numbers…

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production clusters</td>
<td>&gt; 65</td>
</tr>
<tr>
<td>Production nodes</td>
<td>&gt; 2000</td>
</tr>
<tr>
<td>Multi-region clusters</td>
<td>&gt; 40</td>
</tr>
<tr>
<td>Most regions used</td>
<td>4 (three clusters)</td>
</tr>
<tr>
<td>Total data</td>
<td>300 TB</td>
</tr>
<tr>
<td>Largest cluster size</td>
<td>144 nodes (three clusters)</td>
</tr>
<tr>
<td>Max reads/writes</td>
<td>300k rps, 1.2m wps</td>
</tr>
</tbody>
</table>
Example 1

- Subscriber data
- Wide row implementation
  - Row key = customer id
  - Column for each attribute about a subscriber
    - id, name, subscription plan details, holds, …
Example 2

- Movie Ratings
- New ratings initially stored flat (per subscriber)
- Recurring job to aggregate into JSON blob
- Reads grab JSON blob + new writes
Example 3

- Edge Services scripts
  - Versioned, executable Groovy scripts
  - Inefficient query read _entire_ table
  - Created inverted index to get IDs
    - Roll-your-own indices in Cassandra
Life in the cloud
AWS is our home
- Availability Zones (AZ)
- Regions
All Cassandra clusters run in multiple AZs within each region
Some clusters on multiple regions
Resiliency

- We use Cassandra replication factor = 3 / region
- Stripe data across AZs

<table>
<thead>
<tr>
<th>Availability Zone 1</th>
<th>1</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability Zone 2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Availability Zone 3</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>
- Co-process that runs next to Cassandra on every node
- Manages
  - Backup/restore
  - Cassandra bootstrap / token assignment
  - Centralized configuration management
- github.com/netflix/priam
Assigns tokens to nodes at instance launch
Stores token->node assignments in external datastore
  - Reference implementation uses SimpleDB
  - Netflix uses another Cassandra cluster
Stripes neighbor node token in different availability zones
Supports multi-region Cassandra deployments
Automate Cassandra snapshots
- Nightly backups

Copy all Cassandra artifacts to S3
- Snappy compression
- Throttled, multi-part upload
- Data imported to our BI system (Aegisthus)

Restore (full or partial)
- Common for prod -> test refresh
Secondary backup

- Stand alone app for disaster recovery
- Copies all previous day’s artifacts:
  - Secondary Amazon account
  - Secondary cloud provider
- Longer TTL than primary backup
Java client libraries

- Hector
  - [https://github.com/hector-client/hector](https://github.com/hector-client/hector)

- Astyanax
  - Developed by Netflix
  - [https://github.com/netflix/astyanax](https://github.com/netflix/astyanax)

- Cassandra Java Driver
  - Developed by DataStax
  - [https://github.com/datastax/java-driver](https://github.com/datastax/java-driver)
Astyanax

- Clean object model
- Cassandra node discovery
- Node quarantine
- Request failover/retry
- JMX monitoring
- Connection pooling
- Futures execution for timing out long-running queries
Astyanax, Round Robin pool

- Round Robin uses coordinator node to determine a node with data
Astyanax, Token-Aware pool

- Token aware knows where the data resides
- Great for point reads

AB service
astyanax
Challenges of running Cassandra in the cloud
Building for/Deploying in AWS

- Every application change is a new:
  - Build artifact (war, tar, library)
  - AMI (Amazon Machine Image)
  - Auto Scale Group
  - Instance Launch

- No hand-modified configurations in production
… but Cassandra is different

- We can’t drop/launch new instances endlessly
  - We have state!
- Updates are done in place on existing instances
  - Bootstrapping new nodes not free
- We turn over instances at a slow rate
Misbehaving nodes

- Node death happens
  - Just ‘disappears’
  - Disk failure
- EBS wonkiness
  - Currently, root mount is EBS-mounted
Network flapping causes Cassandra to see peers as UP, DOWN, UP, DOWN, UP …. 

Causes

- Weird latency statistics
- Request failure if partitioning is really bad
- Anti-entropy (hints)
- This is good – system as a whole did not fail
SSDs in the cloud!

Welp, not sooo fast

- Higher IOPs, better latencies
- Not quite like SSD on bare metal
BI integration

- Didn’t want BI to run monster distributed queries on prod clusters
- Instead, they:
  - grab the nightly backups
  - Export to JSON
  - Import to Hadoop
  - Find new data
  - Import to Hive, Teradata, etc.
Developer education

- Data model re-think
  - Denormalization
  - Client query tuning is different
  - Eventual consistency
Takeaways

- Netflix is making all of our components distributed and fault tolerant as we grow domestically and internationally.

- Cassandra is a core piece of our cloud infrastructure.

- Netflix is open sourcing its cloud platform, including Cassandra support (Astyanax, Priam, more to come).
Thank you!