Improving Cassandra Client Load Balancing

Ammar Khaku
Joey Lynch
Ammar Khaku

Senior Software Engineer
Cloud Data Engineering at Netflix

Database clients, Java libraries

https://akhaku.com/
Joey Lynch
Senior Software Engineer
Cloud Data Engineering at Netflix
Cassandra Committer
Database shepherd and data wrangler

https://jolynch.github.io/
Outline

Load Balancing Background

Why Stateful Load Balancing is Special

Proposed Solution - Weighted Least Loaded

Experiments and Real World Results
Goal: Upgrade to Datastax 4

Had some performance issues at scale with LoadBalancer and Throttler.
(Un)Balance The Load

A quick crash course on 
queueing theory
and load balancing
**Best in class implementations**

**HAProxy, Nginx, Envoy**
- Weighted **Round Robin**
- Weighted **Least Connection/Load**
- Weighted **Choice of N** (random/hash)

Netflix gRPC: **Random Choice of 2**

**Google** uses **Random Subsetting** with weighted **Round Robin**

Many DB clients choose **Random**
What to choose?

https://github.com/jolynch/performance-analysis/blob/master/not
ebooks/queueing_theory/load_balancing_analysis.ipynb
What to choose?

Response Distribution M/G Process (10000 QPS @ 0.4ms avg with 16 servers):

- weighted-choice-8
- MGk
- shortest-queue
- choice-of-two
- round-robin
- random

Information Free "Static"

What to choose?

Response Distribution M/G Process (10000 QPS @ 0.4ms avg with 16 servers):

- weighted-choice-8
- MGk
- shortest-queue
- choice-of-two
- round-robin
- random

"Dynamic" based on local knowledge

What to choose?

Response Distribution M/G Process (10000 QPS @ 0.4ms avg with 16 servers):

- weighted-choice-8
- MGk
- shortest-queue
- choice-of-two
- round-robin
- random

"Dynamic" based on global knowledge

What to choose?

HAProxy recommends least connections as being strictly dominate to choice of 2 with an efficient impl.

This matches the math and literature absent information.

Google allows servers to communicate back with clients to adjust weights in RR. Very clever.
Stateful Load Balancing

State makes the problem different
The node you hit matters!

- Postgres: master, replica
- ZooKeeper: leader, followers
- CockroachDB: lease holder
What makes Cassandra special?

1. For any piece of data we typically have one replica per availability zone.

2. Depending on the consistency we may need to hop to more hosts.

3. Datastores have hiccups frequently (drives mostly).

4. Our network latency is asymmetric.
Stateful load balancing with real networks

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td><strong>150us</strong></td>
<td>800us</td>
<td>250us</td>
</tr>
<tr>
<td>B</td>
<td>800us</td>
<td><strong>220us</strong></td>
<td>850us</td>
</tr>
<tr>
<td>C</td>
<td>380us</td>
<td>700us</td>
<td><strong>160us</strong></td>
</tr>
</tbody>
</table>
DataStax Java Driver
for Apache Cassandra®
No Token? Round Robin

Token Aware? Hash key, shuffle replicas*, return first. (random subsetting)
Slow to react to slow coordinators, erroring coordinators, paused coordinators, etc …

Traffic often goes cross-zone
No Token?
Round Robin

Token Aware?
Hash key, shuffle replicas, return least loaded between first and second.

Avoids very slow replicas!

Basically choice of 2 over random subsets! Nice!
DataStax Java Driver 4.x for Apache Cassandra®

- Drop in median write latencies
- Drop in 95th, 99th write latencies
- Drop in median read latencies
- Drop in 95th, 99th read latencies
Perf regression with high-throughput cases

We needed to do 20k QPS per client to Cassandra and Datastax 4.x could barely do 8k.
DataStax Java
Driver 4.x for
Apache
Cassandra®

Pays expensive
compare and update
and a lock acquire-release

lock in
throttler

compare-and-swap in
load balancer
Pays expensive **compare and update** and a **lock acquire-release**
Weighted Least Loaded

Started with fixing compare-and-swap, ended up rewriting the algorithm
No Token?
Chose 8 random nodes

Token Aware?
Choose all RF replicas and 8-RF random

Weight concurrency by:
!Rack = 4
!Replica = 12
Unhealthy = 64

Sort the sublist. Done!
Stateful load balancing with real networks

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>150us</td>
<td>800us</td>
<td>250us</td>
</tr>
<tr>
<td>B</td>
<td>800us</td>
<td>220us</td>
<td>850us</td>
</tr>
<tr>
<td>C</td>
<td>380us</td>
<td>700us</td>
<td>160us</td>
</tr>
</tbody>
</table>
set(x=0)
replicas(x) = (s1, s5, s3)

End to End Latency = Latency (L) + Processing (R)

\[ \text{E}_\text{LO} = \frac{1}{3} (L(A, A) + R) + \frac{1}{3} (L(A, B) + R) + \frac{1}{3} (L(A, C) + R) \]

Let \( R = 100 \text{us} \)
\[ \text{E}_\text{LO} = \frac{1}{3} (150 + 100) + \frac{1}{3} (800 + 100) + \frac{1}{3} (250 + 100) = 500 \text{us} \]
**LOCAL_ONE (WLLLB)**

```latex
c1 \rightarrow s1, s5, s3
```

End to End Latency = Latency (L) + Processing (R)

\[ E_{LO} = L(A, A) + R \]

Let \( R = 100 \text{us} \)

\[ E_{LO} = 150 + 100 = 250 \text{us} \quad (50\% \text{ reduction}) \]
set(x=0)
replicas(x) = (s1, s5, s3)

\[ E_{LQ} = \frac{1}{3} (L(A,A) + \min(R, L(A,C) + R)) \]
\[ \frac{1}{3} (L(A,B) + \min(R, L(B,A) + R)) \]
\[ \frac{1}{3} (L(A,C) + \min(R, L(C,A) + R)) \]

Let \( R = 100\, \text{us} \)
\[ E_{LQ} = \frac{1}{3} (150 + 350) + \frac{1}{3} (800 + 900) + \frac{1}{3} (250 + 480) = 980\, \text{us} \]
LOCAL_QUORUM
(Control)

set(x=0)
replicas(x) = (s1, s5, s3)

\[
E_{LQ} = \frac{1}{3} (L(A, A) + \min(R, L(A, C) + R)) \\
\frac{1}{3} (L(A, B) + \min(R, L(B, A) + R)) \\
\frac{1}{3} (L(A, C) + \min(R, L(C, A) + R))
\]

Let \( R = 100 \text{us} \)
\[
E_{LQ} = \frac{1}{3} (150 + 350) + \frac{1}{3} (800 + 900) + \frac{1}{3} (250 + 480) = 980\text{us}
\]
**LOCAL QUORUM (WLLLBB)**

set(x=0)
replicas(x) = (s1, s5, s3)

\[ E_{LQ} = L(A, A) + \min(R, L(A, C) + R) \]

Let \( R = 100 \text{us} \)
\[ E_{LQ} = 150 + 100 + 250 = 500 \text{us} \textit{(50% reduction)} \]
set(x=0)
replicas(x) = (s1, s5, s3)

E_LQ = L(A, A) + \text{min}(R, L(A, C) + R)

Let R = 100\text{us}
E_LQ = 150 + 100 + 250 = 500\text{us} (50\% \text{ reduction})
Experiments
Synthetic Traffic
Latency results
LOCAL_ONE

Median Read Latency
Control: 800us
WLLLB: 500us
Latency results
LOCAL_ONE

Median Write Latency
Control: 800us
WLLL: 500us

WLLL: Flat P99
Control: Variable P99
Latency results
LOCAL_ONE

About a 40% improvement
Latency results
LOCAL_QUORUM
Latency results

LOCAL QUORUM
Latency results

LOCAL QUORUM

About a 10% improvement
## Latency results

<table>
<thead>
<tr>
<th></th>
<th>WLLLB P50/P95/P99 Read (ms)</th>
<th>WLLLB P50/P95/P99 Write (ms)</th>
<th>Control P50/P95/P99 Read (ms)</th>
<th>Control P50/P95/P99 Write (ms)</th>
<th>Read Latency Difference</th>
<th>Write Latency Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>LO-1</td>
<td>0.52/1.30/1.92</td>
<td>0.50/1.30/1.41</td>
<td>0.84/1.45/2.14</td>
<td>0.82/1.35/1.59</td>
<td>38%/10%/10%</td>
<td>39%/4%/11%</td>
</tr>
<tr>
<td>LQ-1</td>
<td>1.33/2.42/2.90</td>
<td>1.21/2.15/2.45</td>
<td>1.52/2.25/3.07</td>
<td>1.36/2.06/2.48</td>
<td>12.5/-7.5%/5.6%</td>
<td>11%/-4.3%/1.2%</td>
</tr>
<tr>
<td>LQ-2</td>
<td>1.40/2.56/4.45</td>
<td>1.27/2.08/2.46</td>
<td>1.55/2.32/3.93</td>
<td>1.32/2.03/2.47</td>
<td>10%/-10%/-13%</td>
<td>4%/-5%/-1%</td>
</tr>
</tbody>
</table>

Why the slight P95 regression in LQ? Theories:

1. Load Imbalance due to asymmetric latency
2. Dynamic Endpoint Snitch
Load imbalance
Reads
Load imbalance

Writes
Network Delay

Force packet delay

Measure results
$ sudo tc qdisc show dev eth0

<table>
<thead>
<tr>
<th>qdisc</th>
<th>Root (8005)</th>
<th>qdisc</th>
<th>Parent 8005:4</th>
<th>Limit 10000p</th>
<th>Flow Limit 100p</th>
<th>Buckets 1024</th>
<th>Orphan Mask 1023</th>
<th>Quantum 18030</th>
<th>Initial Quantum 90150</th>
<th>Low Rate Threshold 550Kbit</th>
<th>Refill Delay 40.0ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>mq</td>
<td></td>
<td>fq 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>fq 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>fq 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>fq 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```bash
sudo tc qdisc show dev eth0
```

<table>
<thead>
<tr>
<th>qdisc</th>
<th>Root (8005)</th>
<th>qdisc</th>
<th>Parent 8005:4</th>
<th>Limit 10000p</th>
<th>Flow Limit 100p</th>
<th>Buckets 1024</th>
<th>Orphan Mask 1023</th>
<th>Quantum 18030</th>
<th>Initial Quantum 90150</th>
<th>Low Rate Threshold 550Kbit</th>
<th>Refill Delay 40.0ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>mq</td>
<td></td>
<td>fq 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>fq 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>fq 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>fq 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Linux Traffic Control (tc)!
Netem to the rescue
(tc-netem)

# Server adds 10ms delay
server$ sudo tc qdisc replace dev eth0 root netem delay 10ms

# Client now observes 10ms additional latency on all requests
client$ ping 100...
...
64 bytes from 100...: icmp_seq=525 ttl=64 time=0.215 ms
64 bytes from 100...: icmp_seq=526 ttl=64 time=0.212 ms
# When netem was enabled
64 bytes from 100...: icmp_seq=527 ttl=64 time=10.2 ms
64 bytes from 100...: icmp_seq=528 ttl=64 time=10.2 ms
64 bytes from 100...: icmp_seq=529 ttl=64 time=10.2 ms
64 bytes from 100...: icmp_seq=530 ttl=64 time=10.2 ms

# Now Revert on server
server$ sudo tc qdisc replace dev eth0 root mq
Netem to the rescue
(tc-netem)

# You can also use netem to simulate packet loss, corruption, duplication, reordering and other TCP issues.
# For example you could add a distribution of delay with

$ tc qdisc change dev eth0 root netem delay 10ms 4ms distribution normal
Slow coordinators

WLLL8 shedding load off the affected coordinator
Slow coordinators

Limited latency impact in 3/3 zones
Slow coordinators

1/12 = 8.3% should have been affected

But only 1.5% were
Garbage Collection

Simulate pauses

Measure results
# pause.sh
while [ 1 ]
do
  sudo -u www-data kill -STOP $(pgrep -f CassandraDaemon)
# Duration of pause
  sleep 20
  sudo -u www-data kill -CONT $(pgrep -f CassandraDaemon)
# Interval between pauses
  sleep 30
done
Slow coordinators

Simulate "GC" pause via stopping the Java process.
Slow coordinators

$1/12 = 8.3\%$ should have been affected

But only $.1\%$ were
Real World Results
Service #1 - LOCAL_ONE

P50 1.1ms -> 0.7ms = 36% improvement
P95 1.9ms -> 1.4ms = 26% improvement
Local One workload
Service #1 - LOCAL_ONE

P50 1.2ms -> 0.7ms = **41%** improvement
P95 2.2ms -> 1.7ms = **22%** improvement
Local One workload
Service #2 - LOCAL_QUORUM

P50 2.0ms -> 1.6ms = 20% improvement
P95 2.8ms -> 2.2ms = 22% improvement
LWT (Local Serial) workload
Service #3 - LOCAL_ONE

P50 1.6ms -> 1.2ms = 25% improvement
P99 5.0ms -> 4.2ms = 16% improvement
Local one workload
Service #3 - LOCAL_ONE

Write Avg

P50 1.3ms -> 0.9ms = 31% improvement
P99 6.0ms -> 6.0ms = ~0% improvement
Local one workload
Uneven distribution of requests across zones
At Scale?
Peak Traffic is 5 Million Writes per Second
Peak Traffic is 6 Million Reads per Second
Scale?

**Median Read Latency**

- **Min**: 0.66
- **Max**: 1.36m
- **Avg**: 1.26m
- **Total**: 424m

**Visibility**: 50.8%

**95th/99th Read Latency**

- **95.0**
  - **Max**: 5.067m
  - **Min**: 0.000
  - **Avg**: 4.188m
  - **Last**: 0.000
  - **Tot**: 1.484
  - **Cnt**: 336.000

- **99.0**
  - **Max**: 319.619m
  - **Min**: 0.000
  - **Avg**: 23.018m
  - **Last**: 0.000
  - **Tot**: 5.000
  - **Cnt**: 336.000

... 2 of 2 lines matched filter ...
Scale?
Conclusions
1. Stay in Zone, failover when loaded

2. LO is easier to load balance for than LQ because we control the entire flow (snitch impacts LQ)

3. We can simulate slow coordinators, and protect against them.

WLLLLB is widely deployed at Netflix handling over 10M QPS