C3: Cutting Tail Latency in Cloud Data Stores via Adaptive Replica Selection

Lalith Suresh
(TU Berlin)

with Marco Canini (UCL), Stefan Schmid, Anja Feldmann (TU Berlin)
Tail-lateness matters

One User Request

Tens to Thousands of data accesses
Tail-latency matters

For 100 leaf servers, $99^{\text{th}}$ percentile latency will reflect in 63% of user requests!
Server performance fluctuations are the norm.

- Resource contention
- Skewed access patterns
- Queueing delays
- Background activities
Effectiveness of replica selection in reducing tail latency?
Replica Selection Challenges
Replica Selection Challenges

- Service-time variations

![Diagram showing client requests to servers with varying response times: 4 ms, 5 ms, 30 ms.](Image)
Replica Selection Challenges

- Herd behavior and load oscillations
Impact of Replica Selection in Practice?

Dynamic Snitching

Uses history of read latencies and I/O load for replica selection
Experimental Setup

- Cassandra cluster on Amazon EC2
- 15 nodes, m1.xlarge instances
- Read-heavy workload with YCSB (120 threads)
- 500M 1KB records (larger than memory)
- Zipfian key access pattern
Cassandra Load Profile
Also observed that
99.9th percentile latency ~ 10x median latency
Load Conditioning in our Approach
Adaptive replica selection mechanism that is robust to service time heterogeneity
• Replica Ranking
• Distributed Rate Control
• Replica Ranking
• Distributed Rate Control
\[ \mu^{-1} = 2 \text{ ms} \]

\[ \mu^{-1} = 6 \text{ ms} \]
Balance product of queue-size and service time \( \{ q \cdot \mu^{-1} \} \)
Server-side Feedback

Servers piggyback $\{q_s\}$ and $\{\mu_s^{-1}\}$ in every response
Server-side Feedback

Servers piggyback \( \{q_s\} \) and \( \{\mu_s^{-1}\} \) in every response

- Concurrency compensation
Server-side Feedback

Servers piggyback \( \{q_s\} \) and \( \{\mu_s^{-1}\} \) in every response

- Concurrency compensation

\[
\hat{q}_s = 1 + os_s \cdot w + q_s
\]
Select server with min $\hat{q}_s \cdot \mu_s^{-1}$?
Select server with min $\hat{q}_s \cdot \mu_s^{-1}$? 

- Potentially long queue sizes
- What if a GC pause happens?

- 20 requests
- $\mu^{-1} = 4$ ms
- 200 requests
- $\mu^{-1} = 20$ ms
Penalizing Long Queues

Select server with min $\left( \hat{q}_s \right)^b \cdot \mu_s^{-1}$

35 requests $\iff$ Server $\iff$ $\mu^{-1} = 4 \text{ ms}$

$b = 3$

20 requests $\iff$ Server $\iff$ $\mu^{-1} = 20 \text{ ms}$
C3

- Replica Ranking
- Distributed Rate Control
Need for rate control

Replica ranking insufficient

• Avoid saturating individual servers?

• Non-internal sources of performance fluctuations?
Cubic Rate Control

- Clients adjust sending rates according to cubic function
- If receive rate isn’t increasing further, multiplicatively decrease
Putting everything together

C3 Client

Replica group scheduler

Sort replicas by score

Rate Limiters

1000 req/s

2000 req/s

Server

Server

{ Feedback }
Implementation in Cassandra

Details in the paper!
Evaluation

Amazon EC2

Controlled Testbed

Simulations
Evaluation

Amazon EC2

• 15 node Cassandra cluster
• M1.xlarge
• Workloads generated using YCSB (120 threads)
• Read-heavy, update-heavy, read-only
• 500M 1KB records dataset (larger than memory)
• Compare against Cassandra’s Dynamic Snitching (DS)
99.9th percentile

<table>
<thead>
<tr>
<th>Replica Selection Strategy</th>
<th>Read Heavy</th>
<th>Read Only</th>
<th>Update Heavy</th>
</tr>
</thead>
<tbody>
<tr>
<td>C3</td>
<td>DS</td>
<td>C3</td>
<td>DS</td>
</tr>
</tbody>
</table>

Lower is better
2x – 3x improved 99.9 percentile latencies

Also improves median and mean latencies
2x – 3x improved 99.9 percentile latencies

26% - 43% improved throughput
Takeaway:

C3 does not tradeoff throughput for latency
How does C3 react to dynamic workload changes?

- Begin with 80 read-heavy workload generators
- 40 update-heavy generators join the system after 640s
- Observe latency profile with and without C3
Latency profile degrades gracefully with C3

Takeaway: C3 reacts effectively to dynamic workloads
Summary of other results

- Higher system load
- Skewed record sizes
- SSDs instead of HDDs

- > 3x better 99.9th percentile latency
- 50% higher throughput than with DS
Ongoing work

- Tests at SoundCloud and Spotify
- Stability analysis of C3
- Alternative rate adaptation algorithms
- Token aware Cassandra clients
C3
Replica Ranking + Dist. Rate Control

Summary