Rein: Taming Tail Latency in Key-Value Stores via Multiget Scheduling

WALEED REDA, MARCO CANINI, LALITH SURESH, DEJAN KOSTIC, SEAN BRAITHWAITE

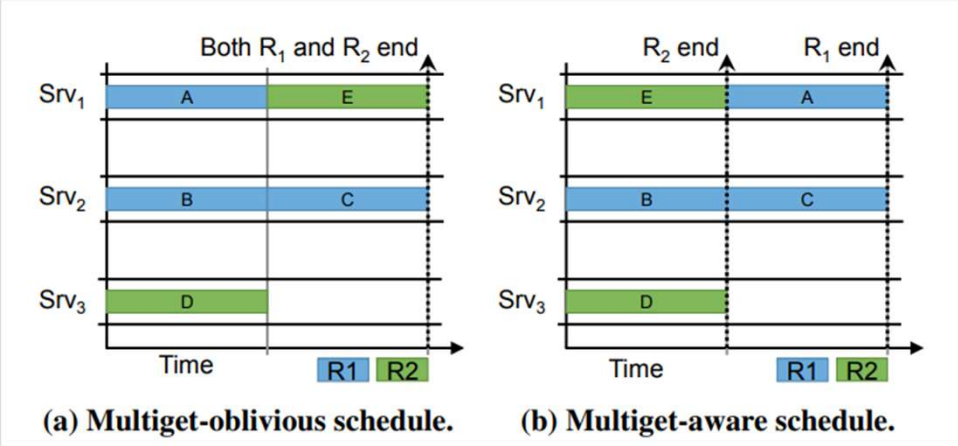
INÊS GARCIA 99083, JOÃO COSTA 99088, MIGUEL MENDO 99111

Introduction

- **Context**: The importance of reducing tail latencies in distributed key-value stores.
- **Problem**: Variability in latency distributions for multiget requests in key-value stores like Cassandra.
- **Objective**: To reduce tail latencies using inter-multiget scheduling.

Motivation

- By profiling SoundCloud CassandraDB, they noticed that high latency sub-requests were withholding the smaller faster sub-requests, consequently bottlenecking the DB's throughput
- They could easily theorize that if they held back the lighter sub-requests until the system could fulfil the heavier ones, it would free resources to reply to other overall less latent requests.



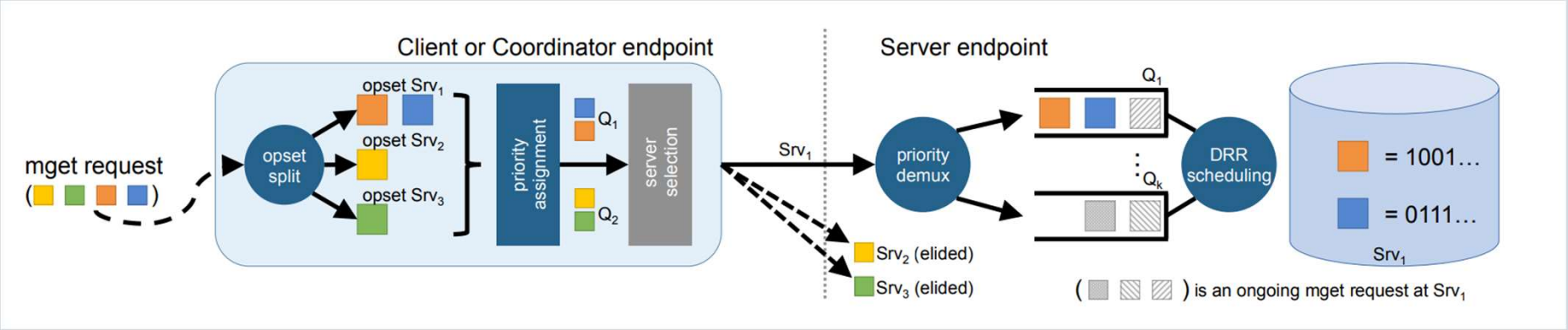
(a) Multiget-oblivious schedule. **(b) Multiget-aware schedule.**

System Design Overview

A system designed to reduce tail latencies via inter-multiget scheduling.

Key Techniques:

- Identifying bottlenecks in multiget requests.
- Scheduling operations to minimize the impact of these bottlenecks.
  - Shortest Bottleneck First (SBF)
  - Slack-Driven Scheduling (SDS)
  - Multi-level queues with Deficit Round Robin (DRR)



Rein specification

- Priority scoring / Bottleneck estimation
  - There are too many variables to consider
  - Metrics won't help because they'll be too late, and they aren't free to calculate
  - There is very low correlation between size of operation and RTT
  - **The solution**: They found out that the number of operations in the request has enough correlation with the RTT, and it is easy to calculate

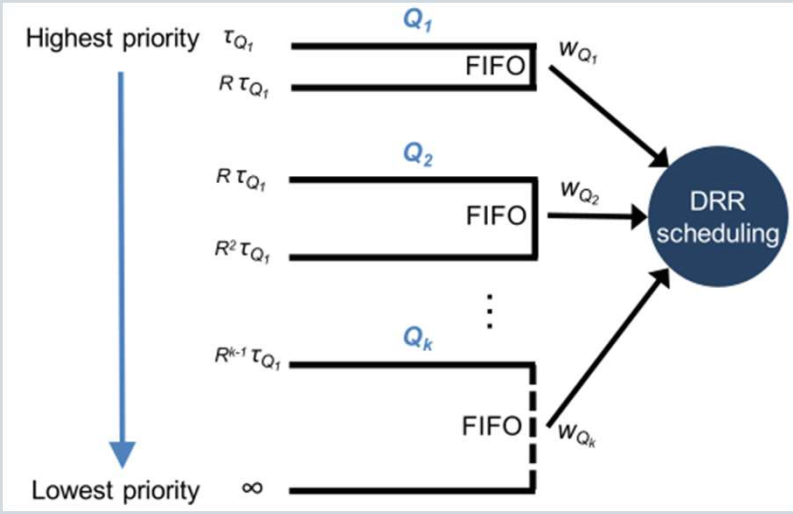
Rein specification

- Shortest Bottleneck First (SBF)
  - Give priority to operations that belong to low bottleneck requests
  - This can cause starvation on heavy requests' side, so waiting time increases the operation's priority
- Slack-Driven Scheduling (SDS)
  - Slack = how much time can it potentially wait for the request's bottleneck
  - Incorporate slack into the priority of the sub-request (x):
    - SDS(x) = (cost(x) + slack(x))/size(x)

Rein specification

- Multi-level queueing
  - DRR-based scheduling
  - Assign the bottleneck of the request to a queue according to its cost
  - Assign the remaining sub-requests to queue with lower priority

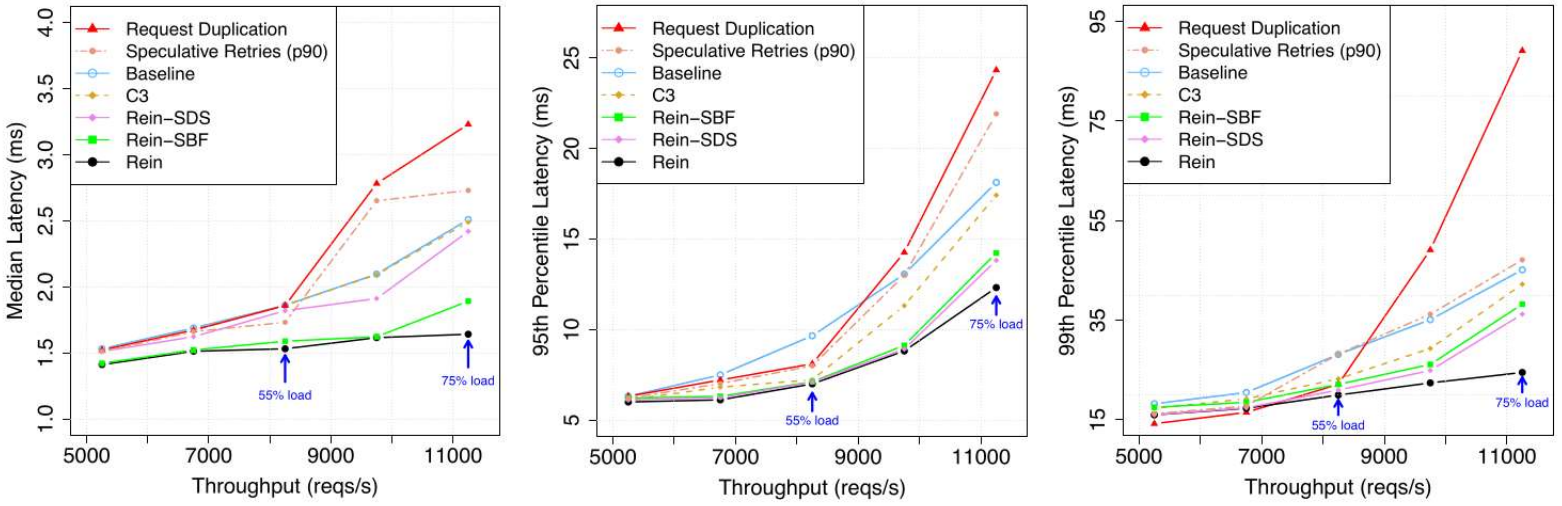
$$Q_{min} = \underset{q \in \mathbb{Q}}{\operatorname{argmin}} \left\| \frac{\text{cost}(op)}{\text{cost}(B)} - \frac{w_q}{w_B} \right\|$$



Evaluation

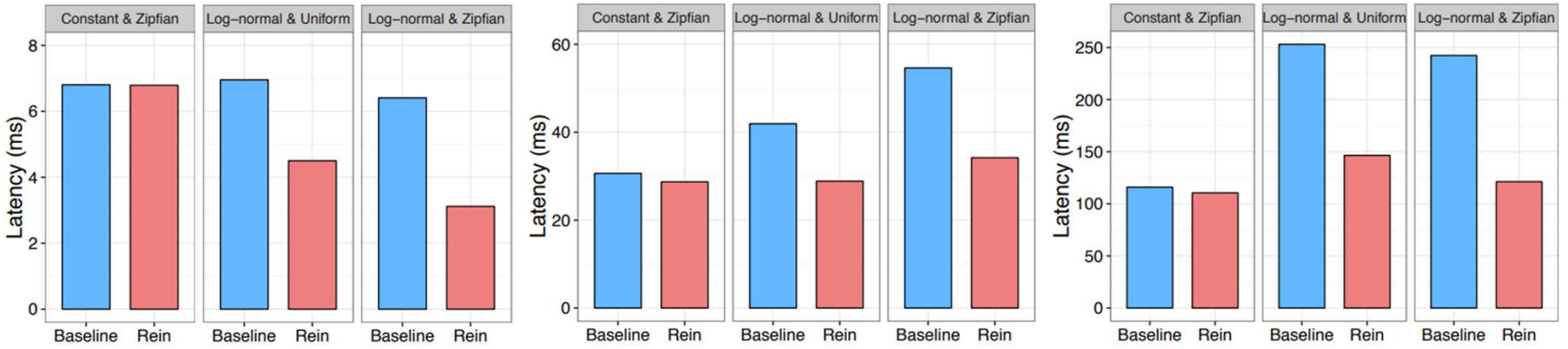
- 16 m3.xlarge AWS EC2 instances: 15 GB of memory, 2x40GB SSD, and 4 vCPUs.
- To generate workloads: modified version of the YCSB, configured it to run on a separate node.
- They inserted rows into Cassandra with value sizes generated following the distribution of Facebook's Memcached deployment
- Replication factor of partitions: 3
- Concurrency level: 8
- Consistency level for all requests: 1
- The automatic paging feature in Cassandra was disabled to make sure the queries were not sent in a sequential manner.

Evaluation



(a) Median latency. (b) 95^{th} percentile. (c) 99^{th} percentile.

Figure 9: Latency attained by the different variants of Rein compared to other latency reduction techniques. The x-axis represents the offered load. We see that Rein's approach achieves the highest gains in the median as well as high percentile latencies.



(a) Median latency.

(b) 95^{th} percentile.

(c) 99^{th} percentile.

Figure 11: Latency comparison of Rein versus the baseline using different synthetic workloads at 75% utilization.

Evaluation

Conclusion

- Investigation of the latency-causing factors
- Assessment of potential for latency reduction
- Design and Implementation of Rein
- Evaluation