RobinHood: Tail Latency-Aware Caching in Large Web Services

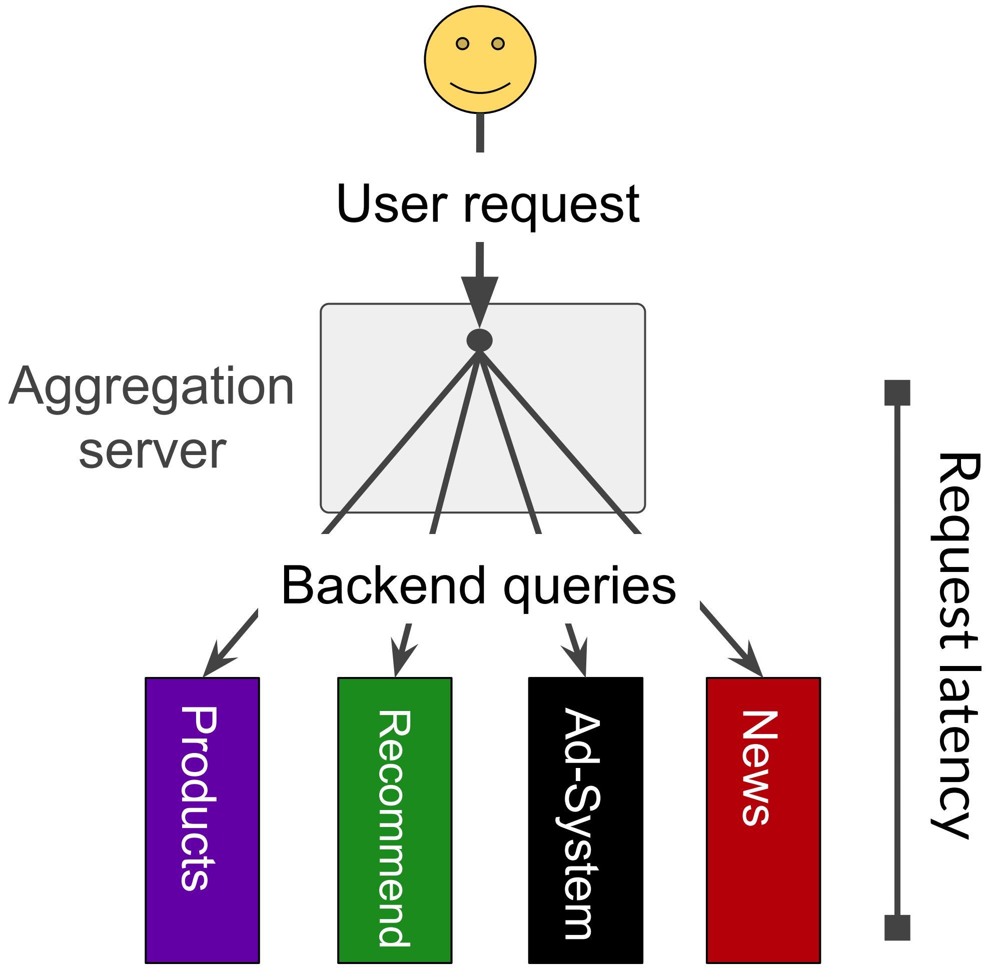
Daniel S. Berger, CMU
Benjamin Berg, CMU
Timothy Zhu, PennState
Siddhartha Sen, Microsoft Research
Mor Harchol-Balter, CMU

To appear at USENIX OSDI (October 2018).

Red Hat Colloquium, 9/5/18,
Hariri Institute for Computing.

Carnegie Mellon University

Microsoft Web Architecture

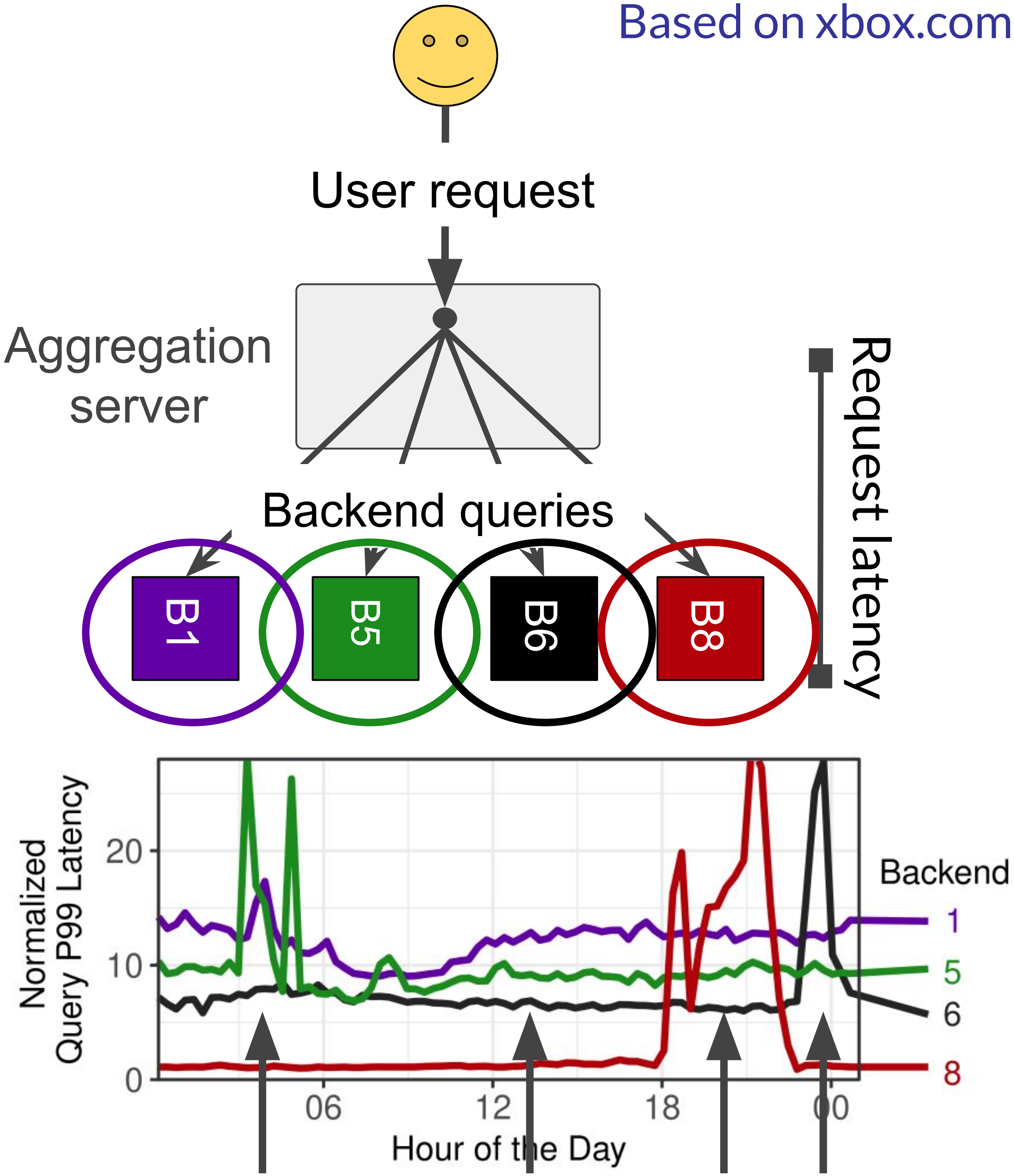


Request must wait for last query!

Goal: minimize 99-th percentile request latency (P99)

What Causes High P99 Request Latency?

Based on xbox.com production trace from 3/2018.

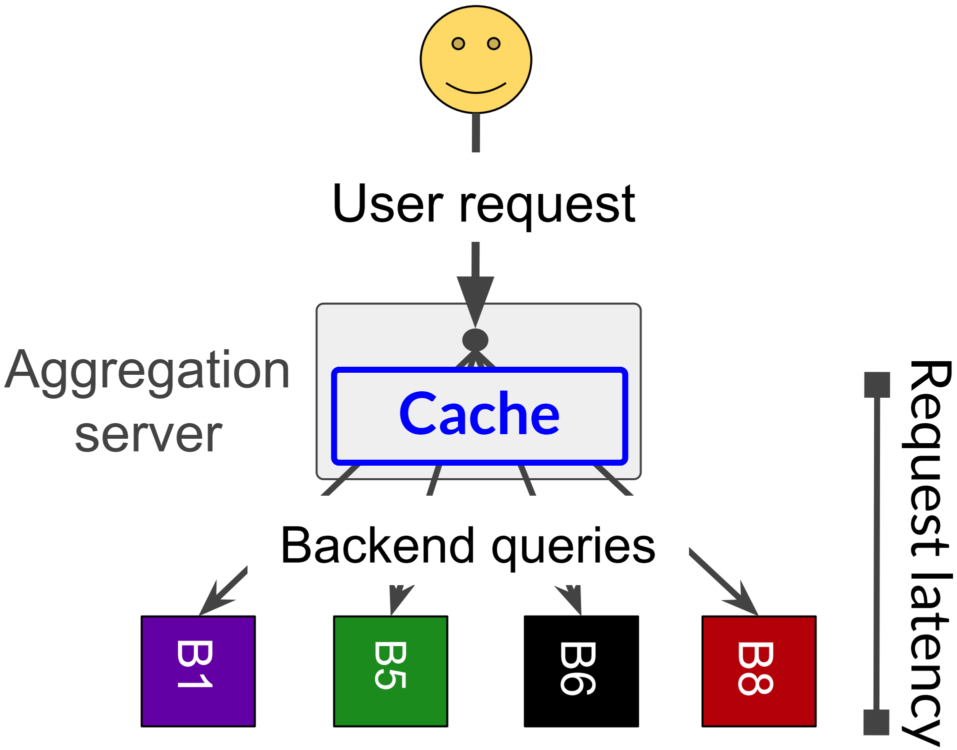


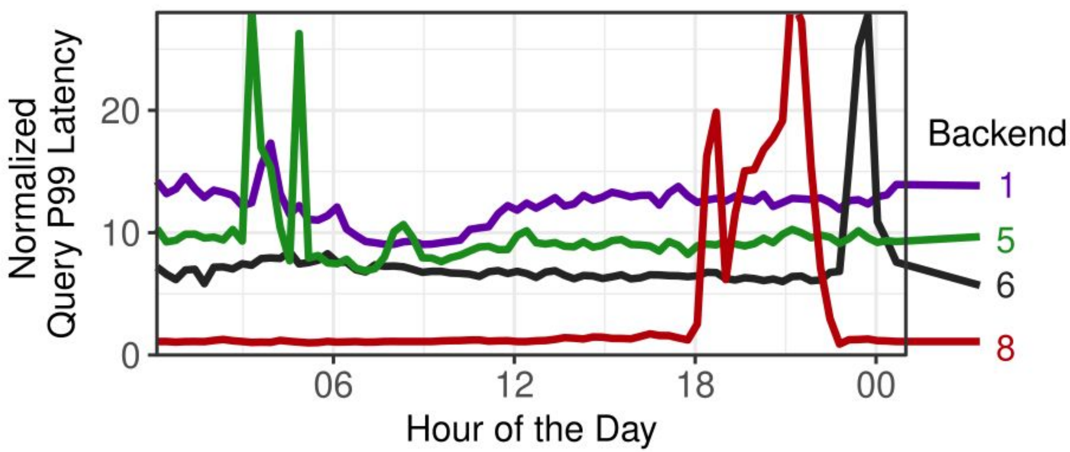
~~Better load balancing?~~

Auto-scaling for backend systems?

already included here

What Else Can We Do?





Aggregation Cache

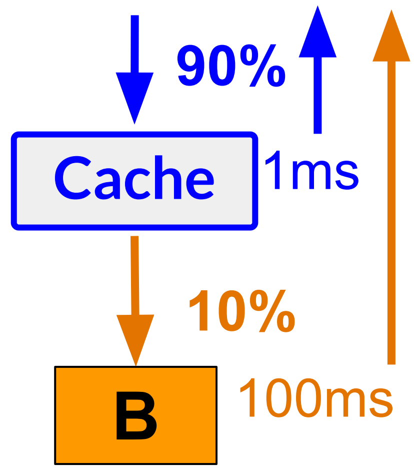
Shared among queries to all backends

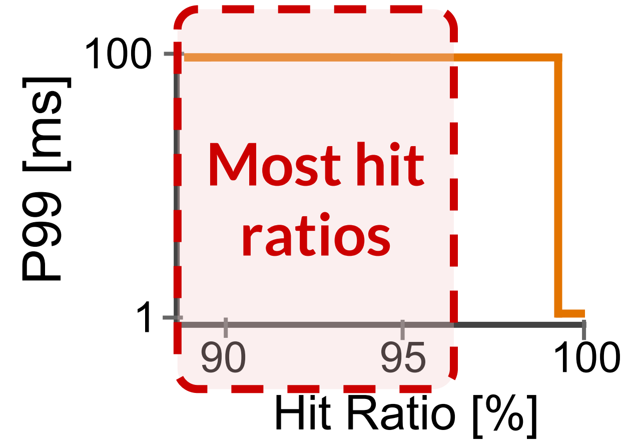
Can we use this cache to reduce P99 request latency?

Can We Use Caching to Reduce the P99?

Belief: **No**

(assuming backend
not overloaded)





BY JEFFREY DEAN AND LUIZ ANDRÉ BARROSO

The Tail at Scale

"The caching layer does not directly address tail latency"

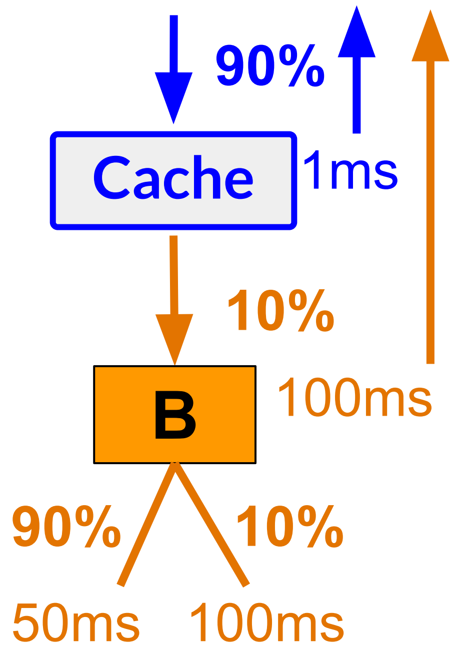
Existing caching systems **do not** attempt to **reduce the P99**

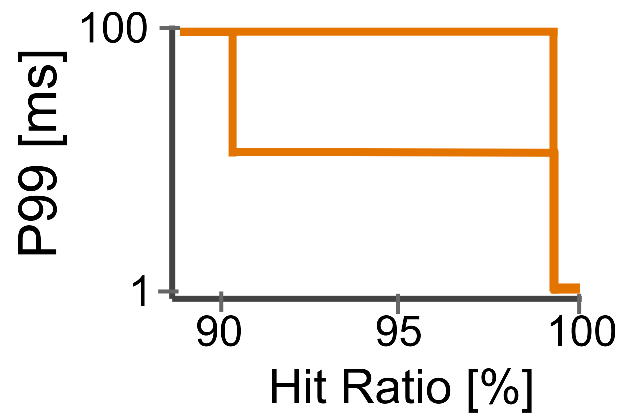
Instead focus on: overall miss ratio or fairness properties

Can We Use Caching to Reduce the P99?

Belief: **No**

(assuming backend not overloaded)





But: **latency is not a constant**

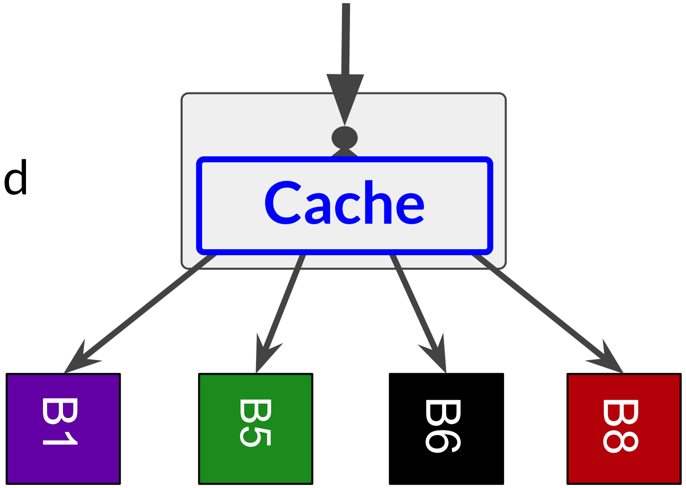
Caching can reduce P99 request latency!

Effectiveness in Microsoft's architecture?

Can We Use Caching to Reduce the P99?

Belief: **No**

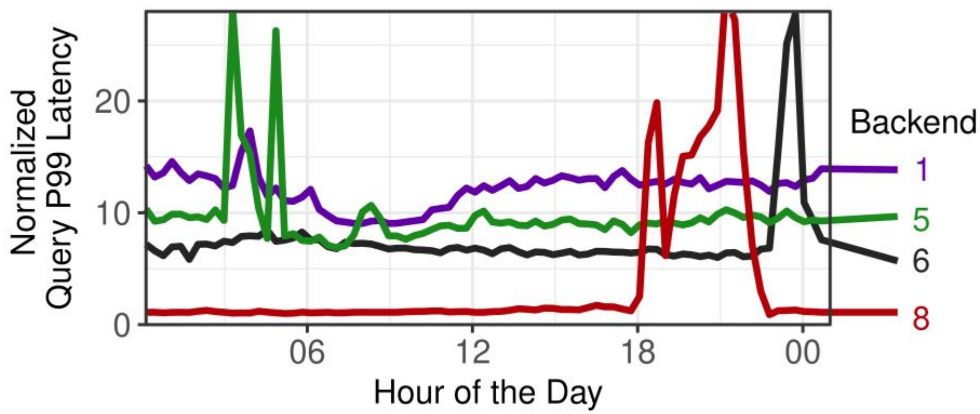
(assuming backend not overloaded)

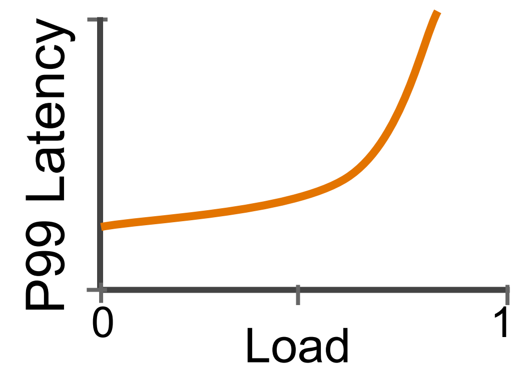


Observations at Microsoft

1) Lower hit ratios, more competition for cache space

2) Some backends are temporarily overloaded





Use the Aggregation Cache as a "Load Balancer"!

RobinHood: Experimental Validation of our "Caching for Tail Latency Idea"

RobinHood Caching System

- Microsoft web architecture
- Partition aggregation cache by backend system
- Minimize request P99 by dynamically adjusting partition sizes

Scalable in #backends, #aggregation servers

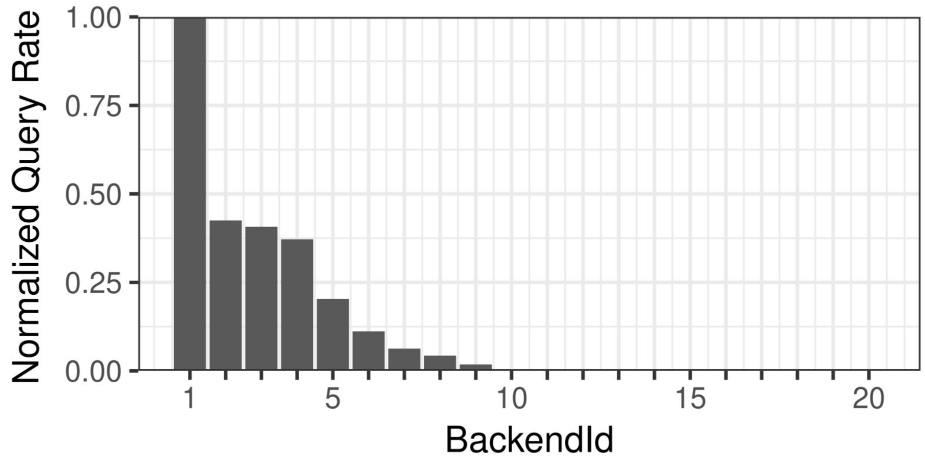
Deployable on off-the-shelf software stack

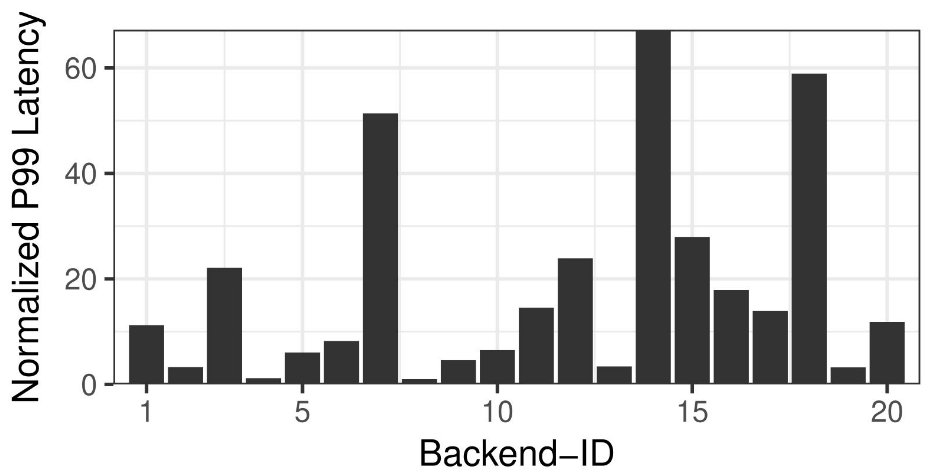
Challenges in Minimizing the Request P99

Use the Aggregation Cache as a "Load Balancer"!

How to define "load"?

1) High Load = High Query Rate



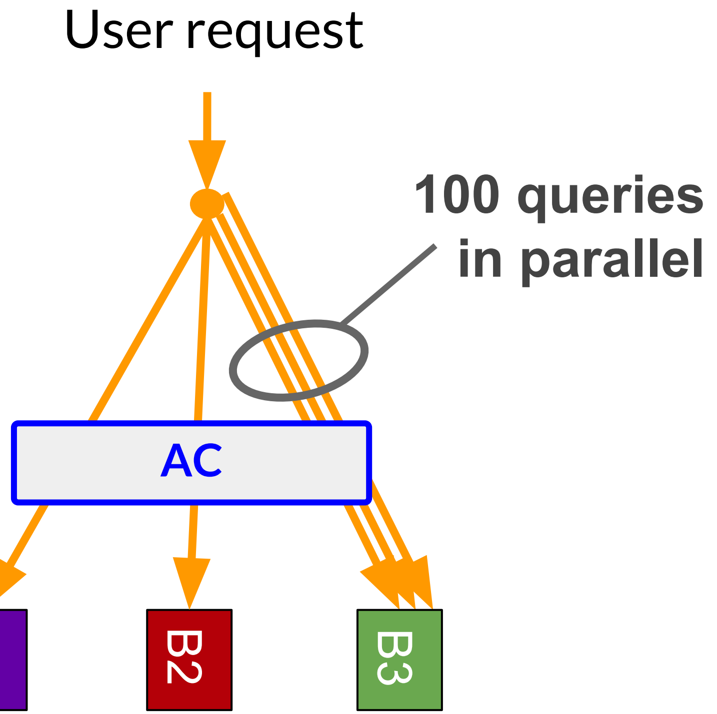


Challenges in Minimizing the Request P99

Use the Aggregation Cache as a “Load Balancer”!

How to define “load”?

2) High Load = High Query P99



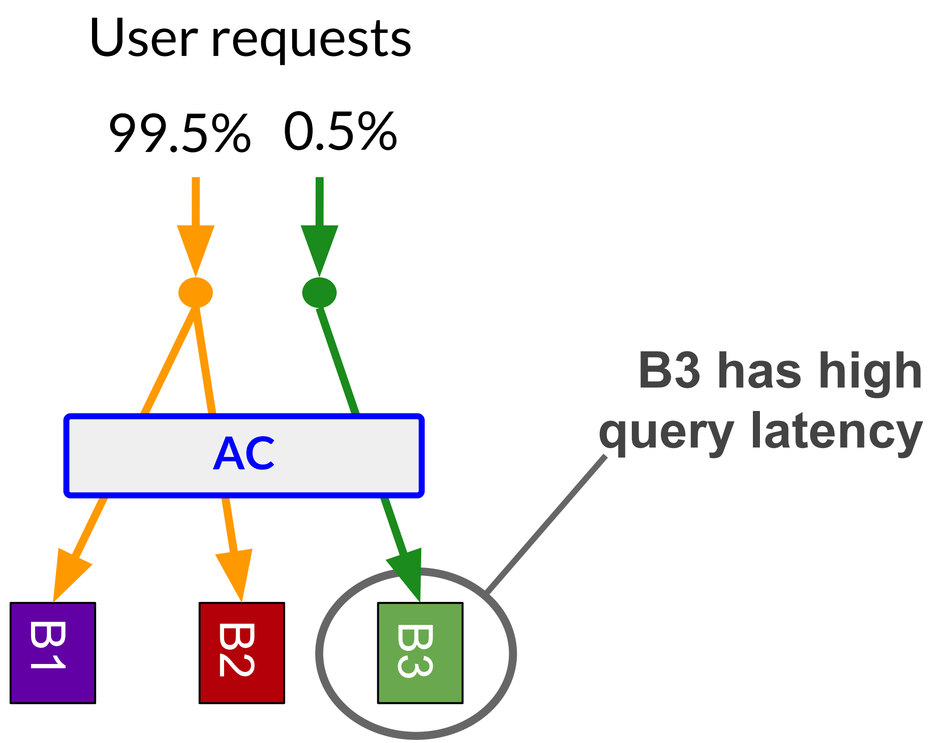
Same P99 on all backends sufficient?

Challenges in Minimizing the Request P99

Use the Aggregation Cache as a “Load Balancer”!

How to define “load”?

3) High Load = High Query Latency (Different Percentiles)



Should we prioritize B3?

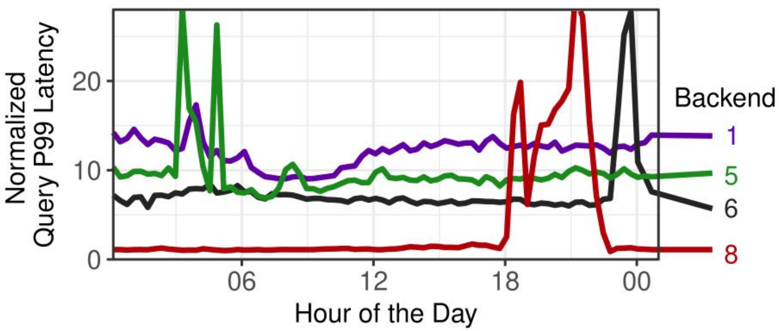
Challenges in Minimizing the Request P99

Use the Aggregation Cache as a “Load Balancer”!

How to define “load”?

Need a new definition of “load”

- Incorporate whether backend “causes” high request P99



- Frequently recalculate load metric

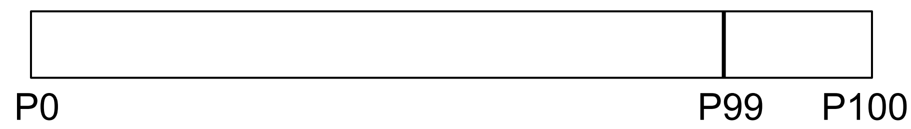
Basic RobinHood Algorithm

Find the backend "causing" high request P99

Challenges:
- Not a single cause
- Sample Variance

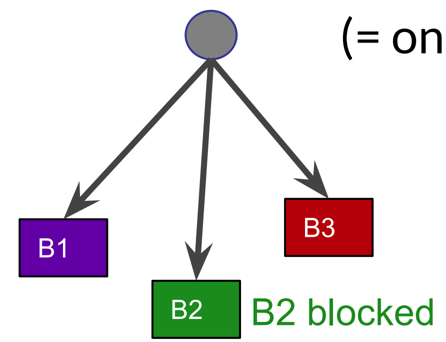
Basic algorithm:

1. Sort all request latencies:



2. Determine who "blocked" P99 request

(= on critical path)



3. Allocate cache space to blocking backend

Refined RobinHood Algorithm

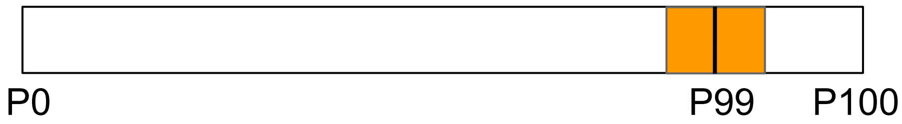
Find the backend "causing" high request P99

Challenges:
- Not a single cause
- Sample Variance

Consider a "neighborhood" of the P99

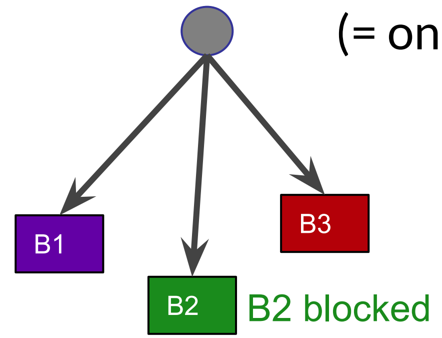
Refined algorithm:

1. Sort all request latencies:



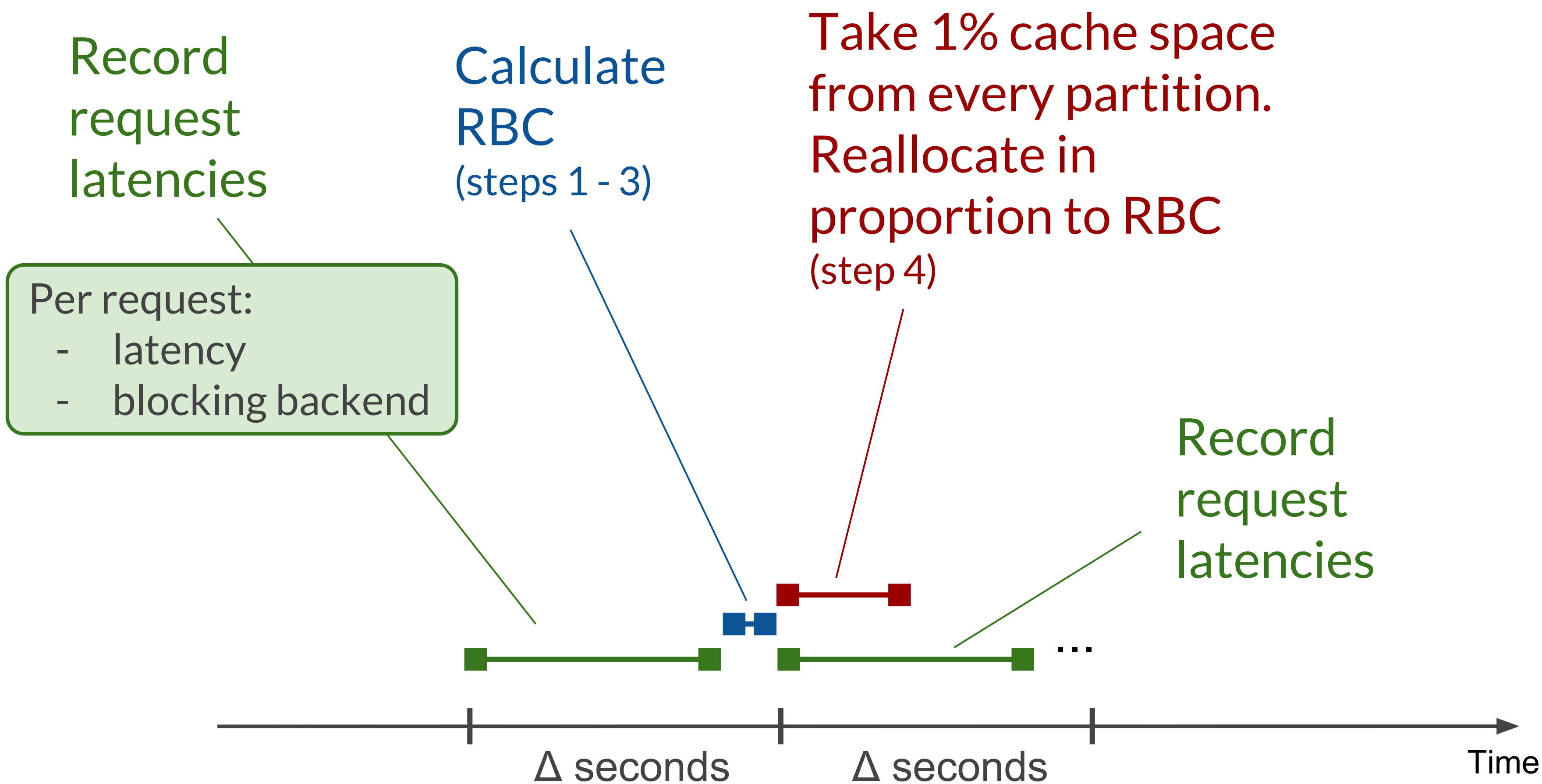
2. X = { requests in P99 neighborhood }
3. Determine who "blocked" requests in X

(= on critical path)

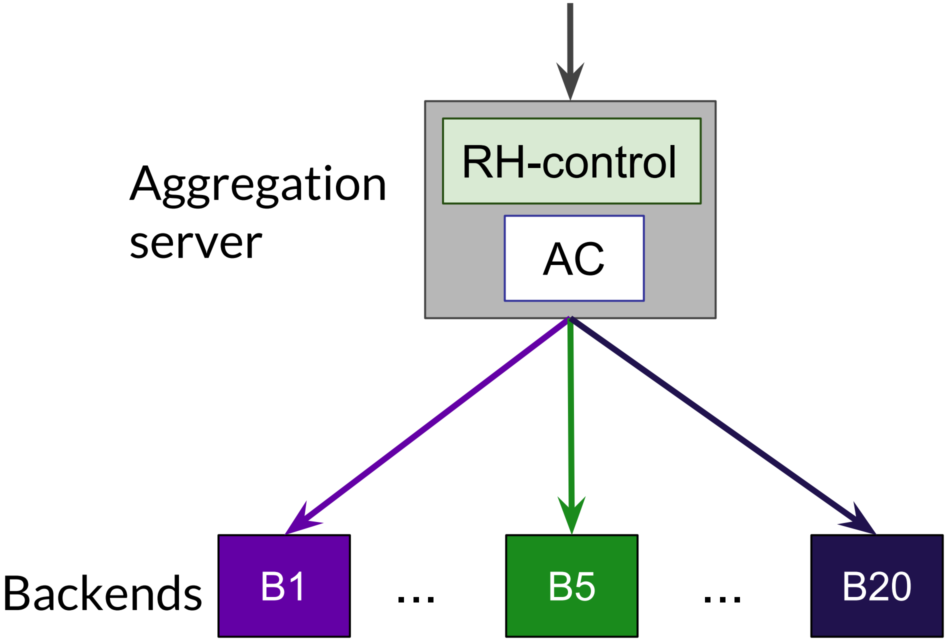


4. Allocate in **proportion** to "request blocking count" (RBC) in X

Dynamic Reallocation with RobinHood



RobinHood Architecture



Aggregation Cache (AC)

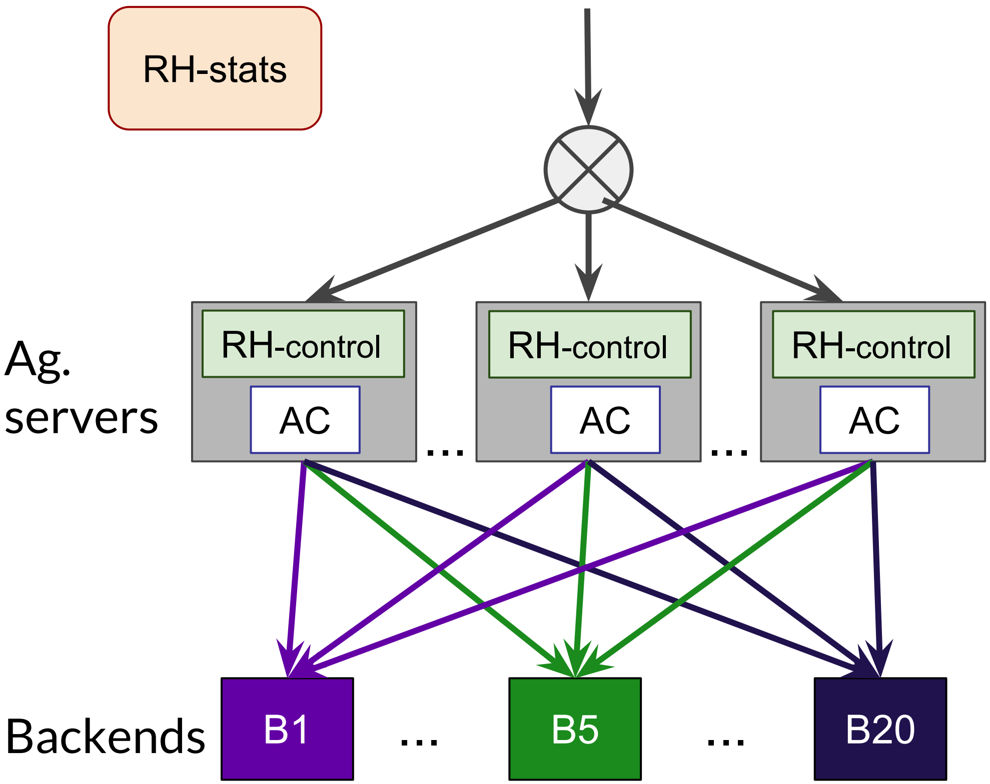
- need support for dynamic resizing
- e.g., off-the-shelf memcached 1.5

RobinHood Controller

- not on request path
- lightweight python
  - computes RBC
  - runs allocation algorithm
  - controls AC partitioning

RobinHood Architecture

Production system: 16-64 Ag. servers



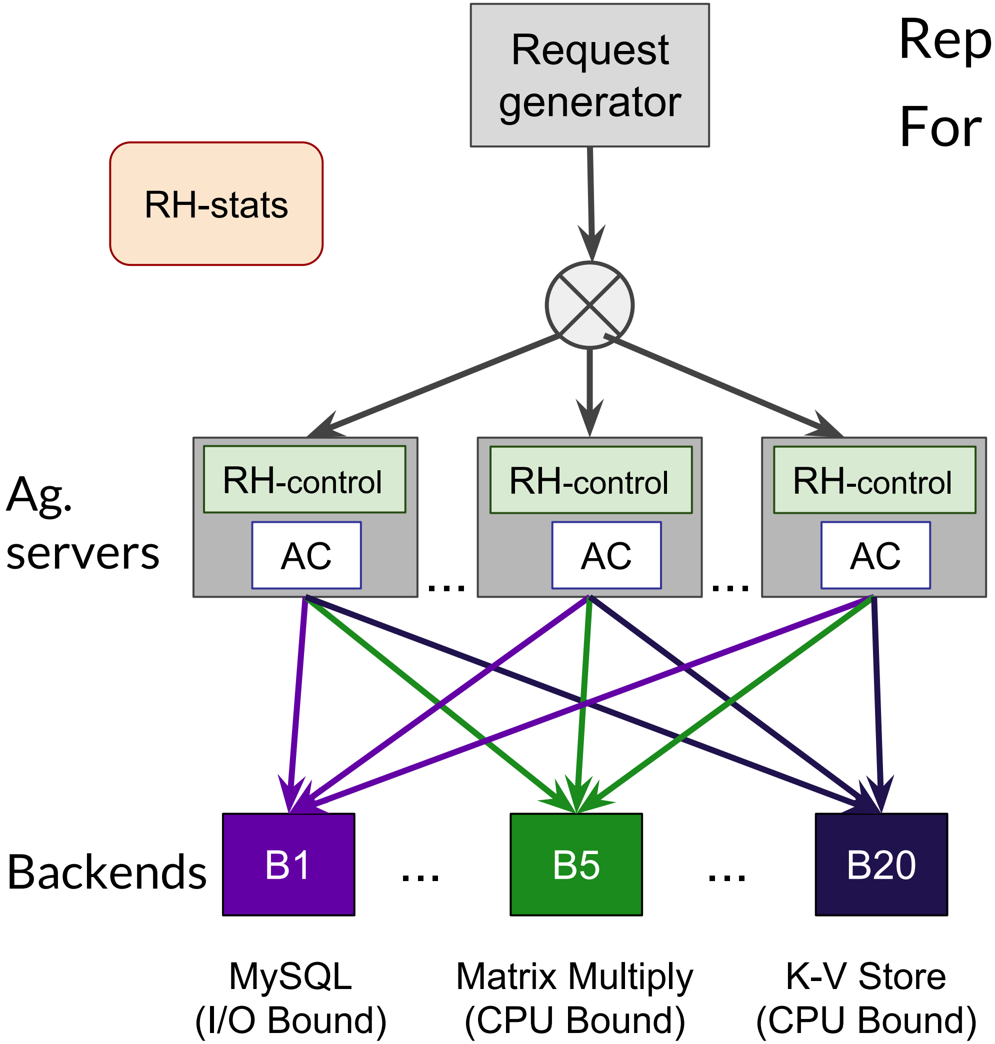
Δ = 5 seconds

⇒ RH-control / AC

Distributed RobinHood:

- Local measurements / Pooled measurements
  - Increase #tail data points
  - Stream to/Pull from central buffer (RH-stats)
  - “Just a buffer” (15s state)
- Local decisions
  - Based on local partition’s allocation speed
  - Transient differences across ag. server

Experimental Setup



Replay production requests and queries

For 4 hours, at 200k queries/second

(max: ~500k queries / second)

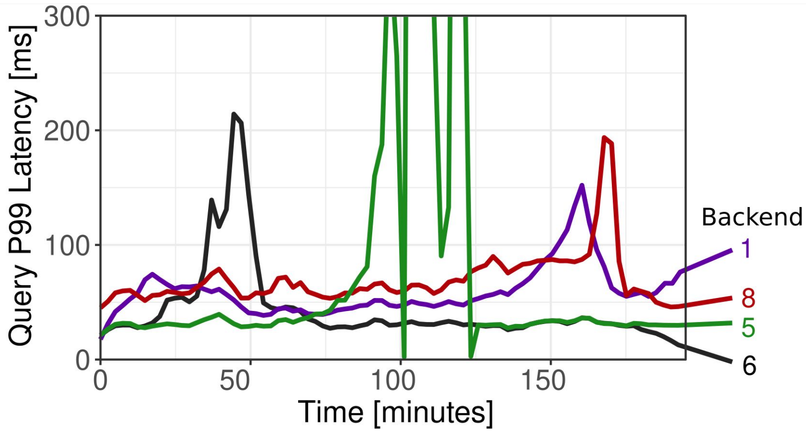
32 GB cache size

16 threads, 8 Gbit/s network

20 backends

up to 8 servers per backend

Emulate query latency spikes



Evaluation Results: P99 Request Latency

RobinHood

[our proposal]

MS Production System

[OneRF]

Minimize overall miss ratio

[Cliffhanger, NSDI'16]

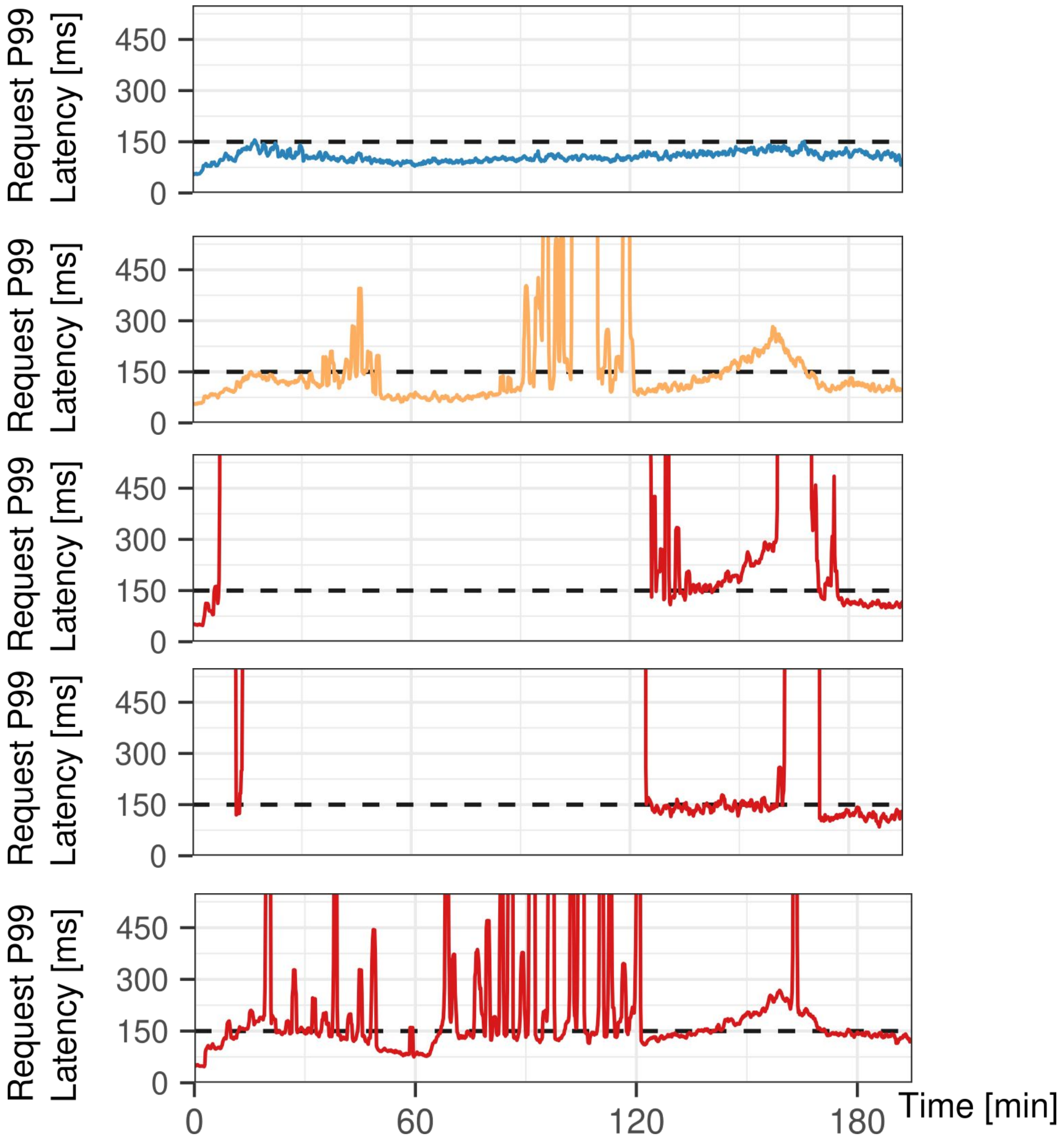
Fairness between partitions

[FairRide, NSDI'16]

Balance query latencies

[Hyberbolic, ATC'17]

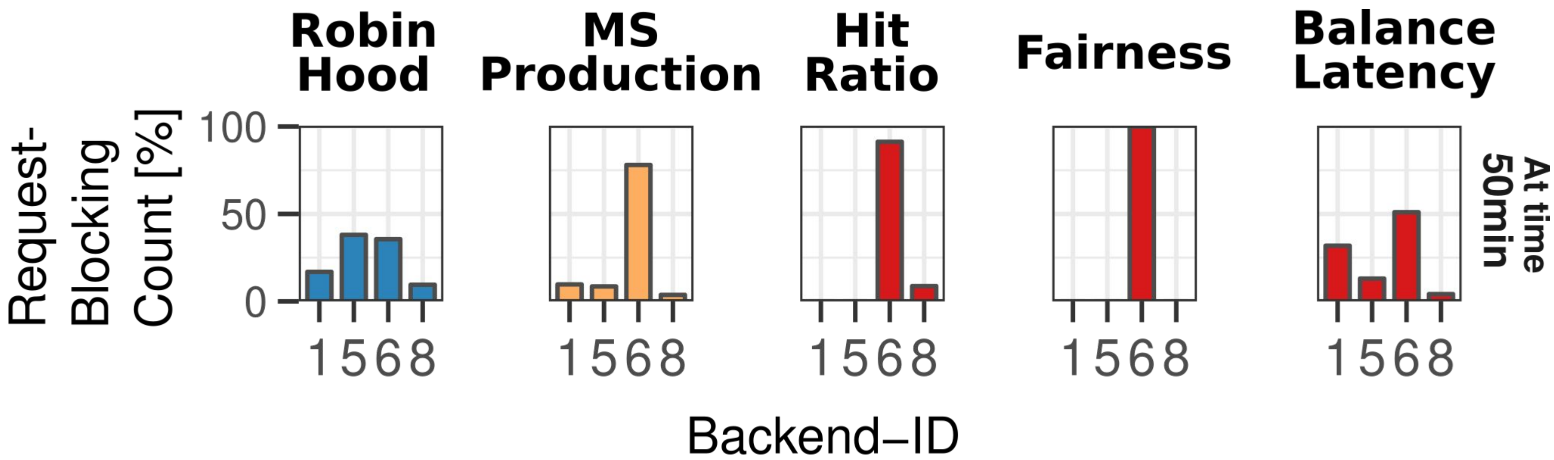
(Improved P99-version)



Evaluation Results: RBC Balance

RBC = request blocking count

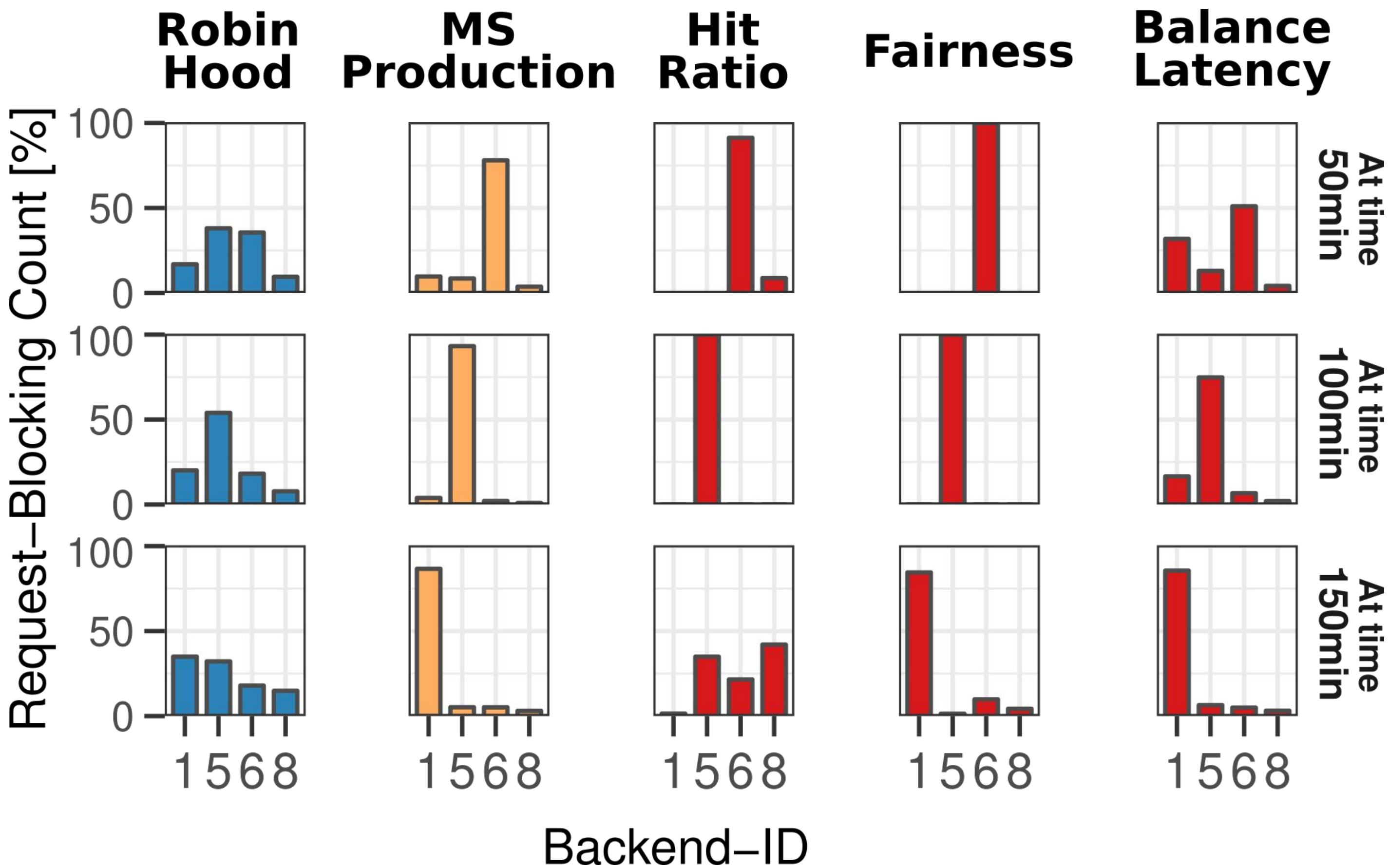
Intuition: balanced ↔ no single bottleneck



Evaluation Results: RBC Balance

RBC = request blocking count

Intuition: balanced ↔ no single bottleneck



Conclusions

Is it possible to use caches to improve the request P99?

Yes! 5x reduction in peak latency, 10x fewer SLO violations.
Caches can be used as load balancers: “RBC load metric”.

Feasibility in production systems?

Yes! Tested on off-the-shelf software statck. Works orthogonally to existing load balancing and auto scaling techniques.

Is this the optimal solution? End of this project?

No! There’s a lot to do, e.g., other types of workloads (Google, FB), other types of systems (apply ideas to resource allocation, …).

Vision: near-optimal allocation based on performance modeling